Polyline Join Evaluation Using Raster Approximation

Rodrigo Salvador Monteiro
Leonardo Guerreiro Azevedo
Geraldo Zimbrão
Jano Moreira de Souza
Computer Science Department
Graduate School of Engineering
Federal University of Rio de Janeiro
PO Box 68511, ZIP code: 21945-970
Rio de Janeiro - Brazil,
Email: salvador@cos.ufrj.br, legaz@cos.ufrj.br,
zimbrao@cos.ufrj.br, jano@cos.ufrj.br

Abstract:

The main subject of spatial joins are polygons and polylines. The processing of spatial joins can be greatly improved by the use of filters that reduce the need for examining the exact geometry of spatial objects in order to find the intersecting ones. Approximations of candidate pairs of spatial objects are examined using such filters. As a result, three possible sets of answers are identified: the positive one, composed of intersecting pairs; the negative one, composed of non-intersecting pairs; and the inconclusive one, composed of the remaining pairs of candidates. To identify all the intersecting pairs of spatial objects with inconclusive answers, it is necessary to have access to their representation so that an exact geometry test can take place. This is true for both polygons and polylines. There are many approximations designed for polygons, but few are suitable for approximating polylines. This article presents a join implementation based on the MSQP architecture [BRIN94] using the Five Color Direction Raster Signature (5CDRS) [ZSMA00] as a filter in the second step, which is polyline approximation. The performance was evaluated with real world data sets. The results showed that our approach, when compared to other approaches presented in the related literature, reduced the inconclusive answers by a factor of more than two. As a result, the need for retrieving the representation of polylines and carrying out exact geometry tests is reduced by a factor of more than two, consequently also reducing the overall time by a factor of more than two, since this step is the most time consuming.

1. Introduction

The field of spatial databases has recently experienced a very fast development. Many systems represent data having spatial attributes. Such systems, known by the denomination of Geographic Information Systems (GIS's), use spatial databases generally constructed upon relational databases, and therefore need specially developed algorithms to meet their specific requirements. As an example, we should mention the efforts that have been made to create a SQL standard for spatial queries [SQL395,SAIF94].

Such systems generally allow us to make spatial queries using some operators similar to those found in relational algebra. Thus, it is extremely important to have an efficient algorithm to perform spatial joins, so that an effective evaluation of the queries can be suitably done. [BRIN94] presents the definition of a modular spatial join processor. This model has been frequently cited in many subsequent works. For this reason this work has been developed in such a way to be fully compatible with that processor.

The efforts that have been made are concentrated in the elaboration of efficient algorithms to perform spatial joins of polygons (regions), such as [ZIMB98]. However, the developed techniques make use of approximations (filters) that may not be suitable for polylines. In [ZSMA00] a raster approximation especially designed for polylines is presented. This can be used for processing either polyline join or polyline-polygon join. In particular, this

work just presents the results of polyline join. The polyline-polygon join is future work. The results were promising, since a reduction of more then 50% in the number of intersection tests was obtained.

The work has been divided in sections, as follows. Section One is an Introduction. Section Two defines the problem itself. Section Three surveys the related literature. In Section Four, we present our implementation of the 3-step architecture, as described in section three. Section Four is divided into three sub-sections, each one devoted to present an architecture's step in more details. Section Five, shows the obtained results. Section Six is dedicated to our conclusions, and this work's contributions. Finally, Section Seven lists the references we used.

2. Defining the problem

As mentioned in [ZIMB98], a growing amount of all the research work on databases deals with spatial databases. Although several questions are still open, there is a consensus about a couple of the requirements that such databases must meet: for example, a spatial database must offer to spatial data at least the same facilities that a relational database offers to conventional data. Therefore, a spatial database must give support to ad-hoc queries involving stored spatial attributes. As in relational databases, such queries must be decomposed in smaller, simpler queries, which can be implemented using a small set of spatial operators.

Considering points, polylines and polygons, the three types most common in spatial databases, there are nine classes of different spatial joins. For it's usefulness and complexity, the polygon join has been the most investigated, while the point join has been the less investigated because of it's similarity with the relational join [SAME90]. However, the polyline join or the polyline-polygon joins are also operations of great utility in spatial databases and also present difficulties similar to the polygon join.

In this work we will be dealing with polyline joins, where spatial objects are characterized by having at least one attribute that describes its spatial extension by means of one or more polygonal lines. The only restriction made is that the object must not occupy area.

2.1 Spatial joins

We can view the spatial join as a subset of the Cartesian product of two sets, A and B, not necessarily distinct, containing, respectively, m and n elements [ZIMB97]. This resulting subset is composed of the elements that meet a given spatial predicate. The overlap of spatial objects is of special interest in practical applications. Therefore, this work is concerned solely with the join of intersecting polylines.

Spatial joins may be more efficiently evaluated by means of indices previously built on each data set. The joins simultaneously traverse these indices, searching for object intersections. This approach corresponds to a sort-merge approach in relational databases, and is pointed in [BRIN93] as being quite efficient, especially when the indices already exist. Typically, an index is composed of two parts: the index structure, which only stores the data keys, and the data structure, which stores the data itself. Thus, a spatial index should store the objects in a spatial structure according to a geometric key [BRIN94]. Due to its simplicity, MBR (minimum bounding rectangle) is the most popular geometric key. When we use MBRs, the complexity of a spatial object is reduced to four parameters that retain the most important characteristics of the object: position and extension. Nonetheless, as pointed in [BRIN94] e [ZIMB98], objects of the

real world are very poorly approximated by MBRs – in particular, polylines are very badly approximated by MBRs.

Finally, it is important to note that a simpler approach would consist of inserting each segment of each polyline in an R-Tree and perform the spatial join in this structure. In practice this approach is not feasible in the case where the number of segments in each polyline is big when compared to the number of polylines – in tests presented in [ZSMA00], around forty segments per polyline in sets of 150 polylines was sufficient to make this approach inefficient. This happens for the following reasons: the R-Trees generated have a high superposition degree; contain many elements resulting in trees with height larger than the trees of just polyline MBRs; and finally as each polyline has many segments, if we are interested in obtaining as a response set just the polyline's identities we will have to filter possible duplicated pairs.

3. Related works

In order to improve the efficiency, organization and study of indices and filters for spatial data, Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider e Bernard Seeger proposed in [BRIN94] a three-step architecture for spatial join processing (figure 1). The main target of this architecture is to accelerate the most costly step by reducing the number of spatial objects left to be compared. Such a reduction is done by applying filters in previous steps. Our work consists of an implementation of this architecture's steps, in order to perform polyline join, using the raster approximation presented in [ZSMA00] in the second step.

The first step of this architecture corresponds to a Spatial Access Method (SAM) which is meant to reduce the search space. The access methods traditionally used make use of the object's Minimum Bounding Rectangle (MBR), therefore, this step does not have as output the result of the join operation, instead, it provides a set of candidate pairs which corresponds to a super-set of the solution. This step is know as MBR join. It is important to note that independently of the method used to implement it, the final product should be the same set, since it is composed by the intersecting MBRs.

The second step consists in comparing the candidate pairs obtained in the previous step using a geometric filter. A geometric filter uses a compact and approximated representation of the object trying to retain its main characteristics. As the result of comparing the filters of two candidates we have three possibilities: the pair belongs to the solution (hit); the pair does not belong to the solution (false hit); and inconclusive.

The input to the third step is the set of object pairs that had an inconclusive geometric filter test. This third step consists in comparing the object pairs through their real representation. This is the most costly step, requiring CPU time to compute the exact intersection, and I/O time to read the spatial objects from disk. The amount of time spent in this step can be considerably reduced with better approximations in the previous steps.

Thus, invariably all the algorithms for spatial join, that have been proposed in the last years, perform a pre-join on the polygon's MBRs, and then investigate which pairs of the candidate set will be reported in the solution. Many of these works stop in this step, do not perform the second and third steps of the MSQP: many just perform the MBR join. Other works do not implement the second step, passing the set produced by the MBR join directly to the third step.

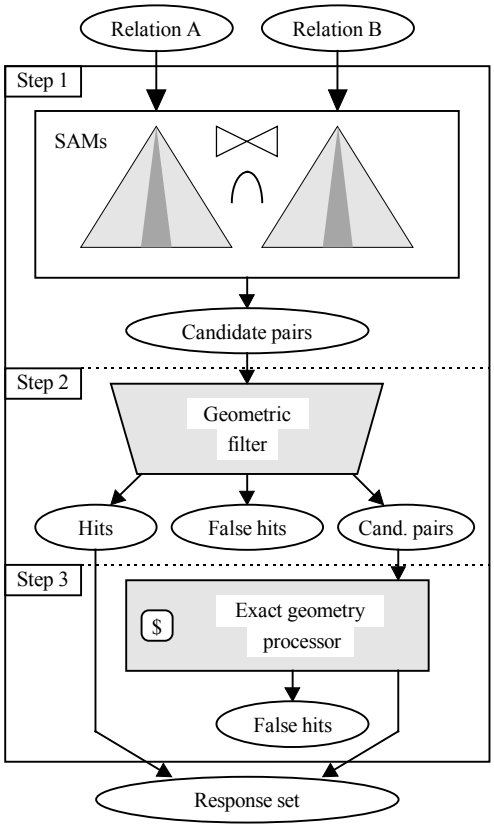

Figure1 – three-step architecture (the Multi-Step Query Processor - MSQP)

As observed before, regardless of the method used to perform the MBR join the result will necessarily be the same set of intersecting MBR pairs. Thus, different methods can be proposed and evaluated for implementing the first and second steps of the MSQP in an independent way. It is sufficient to observe the space requirements for the approximations storage used in step 2. This is why many works just perform the MBR join: improvements in this step will almost immediately result in improvement in the total time spent to perform the join, since this will not affect the performing of the subsequent steps. In fact, the only possible variations in the response set of the MBR join are the order of pair generation, and if we relax a little the definition of a set, the occurrence of repeated pairs.

As far as we know, none of the proposed algorithms to implement step 2 is sensible to the order in which the pairs are generated since the approximations are usually stored in the index. The third step is more sensible to this order, because as polygons in general will not be stored in the index, an investigation of a possible intersection, which was not decided in the previous steps, will be necessary to perform one disk access to retrieve each polygon. This is a potentially costly operation, because it involves one disk access in a random position. In order to minimize the time spent to retrieve the spatial object from disk, we should use cache structures, in other words, we should keep a subset of these objects in main memory avoiding to read the same object several times. It's clear that the order in which the pairs are presented to step 3 will affect cache performance. After all, if an object appears in several pairs, the chance to have a hit in the cache will be higher if all these pairs are passed to step 3 close in time.

Next, we summarize other related works that describe algorithms that can be used to replace any of the steps of the MSQP. We shortly comment some of these works and indicate which step each one of them can replace. This is not a comprehensive list of recently published papers related to this area, but it includes the most important results reached, as far as we know.

Note that just the algorithms related to step 2 (geometric filter) are inadequate to perform polyline joins. For, step 1 presupposes the use of MBRs and the proposed algorithms for step 3 can calculate the exact intersection for both polygons and polylines.

Step 1: In this step, [BRIN93] and [BRIN94] use an R*-tree as a spatial index in order to perform a spatial join. [DEWI96], [MING95], [MING96] and [ZIMB97] present hash based approaches, while [BRIN96] presents parallel algorithms for R*-tree based spatial joins. [BERC96] suggests another tree, the X-tree, which can also be used to implement this step. [HUAN97] proposes a new algorithm to traverse the R*-tree called BFT. [KOUD97] uses an algorithm based on the separation by MBR size in a special tree. [ARGE98] proposes an algorithm based on the well-known plane-sweep algorithm to perform the join when there is no index on the sets. Finally, [MAMO99] proposes a new algorithm to be used when just one of the sets has an index, typical situation that occurs when one query is chained to another.

Step 2: In this step, [BRIN94] uses five-corner polygon approximations, and [BRIN93b] discusses other approximations that can be used to filter the candidate set. [VEEN95] uses approximations that are constructed by rotating two parallel lines around the object. [ESPE97] uses orthogonal polygons to filter some spatial queries. In [ZIMB98] we have an approach using raster approximations. In [ZSMA00] is presented a raster approximation for polylines, which we will approach with more details later. Finally, in [BRIN95] a study on the complexity of polygons is presented, study which is indirectly related to the quality of approximations.

Step 3: In this step, [BRIN94] uses plane-sweep and sets of trapezoids to compute the exact intersection test, and [HUAN97] uses an algorithm called Symbolic Intersect Detection to reduce the time spent on the plane-sweep algorithm. We used an algorithm based in *scan-line* that will be explained later.

The spatial join processor defined by [BRIN94] allows a modularization with no precedents in the performing of spatial joins. Some of the other works referenced also can be adapted to fit the MSQP, implementing some of its steps. An important result obtained by [BRIN94] is that the exact intersection test is the one that consumes more CPU and I/O resources, resulting in the bigger slice of the total spent time. In this way, improvements in the previous steps will be more effective if they also reduce the number of exact intersection tests, in other words, if they reduce the input set to the third step. As step 1 will always produce the same result, independent of the algorithm used to implement it, we can affirm that only improvements in step 2 will reduce the number of exact intersection tests. Next, we will present in detail our three-step architecture implementation in order to perform polyline join.

4. The three-step architecture implementation for polyline joins

4.1 1st Step: R-tree with variable sized entries*

The R*-tree was chosen as the spatial access method to fit the join's first-step. This choice is due to the wide use of this structure, as well as, to the successful results found in literature. However, the R*-tree stores entries with a fixed size which should be established in advance. Since the signature to be stored (5CDRS [ZSMA00]) possesses a size variation not despisable, the adoption of a superior limit for the size of the entry would implicate in a great space waste. It became necessary to implement some adaptations in the R*-tree.

Since the signatures in an R*-tree are only stored in the leaves, the structure and algorithms for the internal nodes are not altered. For the leaf nodes a new structure was derived which allows its entries to have a variable size, but limited to a pre-established interval. This

interval is given by the maximum and minimum sizes of the signature defined in the moment of its creation.

The structural difference of the leaf node implies in adaptation of its algorithms. The split, for instance, instead of being a function of the number of entries becomes a function of the total size of the signatures stored in the node, taking place when this exceeds the node's size.

4.2 2nd Step: 5CDRS (5 Colors Directional Raster Signature)

The 5CDRS filter, presented in [ZSMA00], is a raster approximation for polylines, which extends the 4CRS filter (4 Colors Raster Signature) for polygons approached in [ZIMB98], developed to identify intersection and non-intersection of polylines. For this, it stores the displacement directions of the polyline in a grid. Accompanying the polyline, starting from an extremity, and using the displacements stored, each grid cell has one of the five intersection types depicted in table 1 and illustrated in figure 2 (a).

Cell type	Description
Horizontal and Vertical	Polyline intersects the cell horizontally and vertically
Horizontal	Polyline intersects the cell horizontally
Vertical	Polyline intersects the cell vertically
Inconclusive	Other types of intersection
Empty	Polyline does not intersect the cell

Table 1 – Cell types

For each cell, its intersection type is stored. If the same cell of a signature presents more than one intersection type, the most conclusive type will be stored (figure 2 (b)).

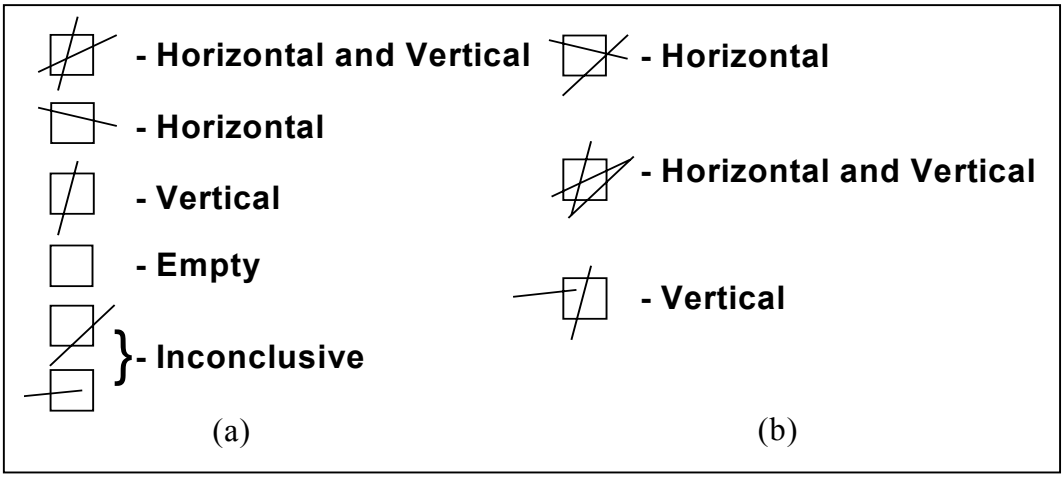

Figure 2 – Cell Types

Table 2 presents the possible cases when the comparison of two cells take place, where discarded and accepted correspond respectively to cells that don't possess and do possess intersection among the two polylines being analyzed. There are 25 possibilities of which 64% are conclusive and 36% inconclusive.

Cell type × Intersection	Empty	Inconclusive	Vertical	Horizontal	Horizontal and Vertical
Empty	**Discarded**	**Discarded**	**Discarded**	**Discarded**	**Discarded**
Inconclusive	**Discarded**	*Inconclusive*	*Inconclusive*	*Inconclusive*	*Inconclusive*
Vertical	**Discarded**	*Inconclusive*	*Inconclusive*	***Accepted***	***Accepted***
Horizontal	**Discarded**	*Inconclusive*	***Accepted***	*Inconclusive*	***Accepted***
Horizontal and Vertical	**Discarded**	*Inconclusive*	***Accepted***	***Accepted***	***Accepted***

Table 2 – Possible cases when comparing a pair of cells

Note that a pair of intersecting cells is sufficient to conclude that the polylines intersect, while it is necessary that all cell pairs be discarded to conclude that they don't intersect. The other cases are inconclusive.

In [ZSMA00] results are presented making clear the efficiency of the 5CDRS filter that reduces about 50% of the candidate pairs that pass from the 2nd to the 3rd step. Even so, we developed signature improvements in the implementation of the MSQP architecture. The signature, in [ZSMA00], is computed for the whole polyline, in other words, the approximation of all parts are stored in a single signature. Using this approach, the MBR's area corresponds to the space of the polyline. In addition the maximum number of vectors per part is reduced as the number of parts increases. This is due to the need for storing each part's header. Intending to increase the precision, we implemented a signature for each part instead of one for the whole polyline. As result, we obtained smaller MBR areas for the signatures. This fact has two direct consequences: many candidates that passed from the 1st to the 2nd step started to be filtered in the 1st; and a larger number of conclusions in the 2nd step was obtained due to the increase in the grid's precision. The second consequence is due to the fact that an inconclusive cell comparison is sufficient to report an inconclusive result, and the precision increase reduces the number of inconclusive comparisons once the polyline is better approximated. The results obtained will demonstrate the facts here emphasized.

4.3 3rd Step: Scan Line

This is the most costly step of the MSQP architecture since the exact representations of the objects are compared, requesting I/O time to read them from disk and CPU time to process the exact intersection. In order to obtain a better performance, an index file was implemented (*hash table*) to direct access the wanted polyline. In this index, each entry stores the polyline's relative position from the beginning of the file. To accelerate this step we implemented an algorithm based on scan line, which is divided in 3 stages. In the first stage, the segments of each polyline part are partitioned in monotonic sequences according to its first point and abscissas/ordinates axis increasing direction. In the following stage, all segments are stored in an increasingly ordered heap according to the abscissas/ordinates axis. The initial comparison values are the first point of each monotonic sequence. In this structure, each segment is associated to the corresponding polyline, part and sequence identifiers. In the last stage, an ordered list of segments is built in the ordinates/abscissas axis increasing direction, opposite to the structures of the previous steps, which is responsible for testing the intersection. The construction of this list is obtained removing the element of greater priority from the heap (root), inserting it at the end of the list or, if it exists, in place of the element belonging to the same sequence. At this time, the intersection of the segment just inserted is tested against all the others in the list. If an intersection with another polyline's segment is detected, then it reports that the polylines intersect. If during

all the execution of the algorithm no segment of different polylines intersects then it is reported that the polylines don't intersect.

5. Experimental Results

We performed a series of tests using real data. The tests meant just to evaluate the efficiency of the filter in order to demonstrate its feasibility when fitting the second step of MSQP architecture. The time spent to read each signature will be embedded in the time spent in step 1, therefore the only times that may influence the join performance as a whole are the time spent in the conversion from vector signature to raster signature (details in [ZSMA00]) and the time spent in the comparison. Such times were calculated and are insignificant (between 2 and 3%) when compared to the time spent in performing the MBR join, since they are just simple integer operations in memory. Our data sets are from the real world, and they correspond to locations of highways and rivers of Canada and part of New York. In both cases a maximum resolution of 350 cells was used for the grid of each signature.

Executing the architecture for Canada's highways and rivers join, the first step resulted in 1016 MBR's comparisons of which 171 overlapped. It should be emphasized that, in this case, there may exist repeated polyline pairs, because the comparison is done among polyline parts and not for the polylines themselves. The inconclusive pairs are passed to the following step. The second step resulted in 3 intersections and 73 non-intersections, in this case, eliminating repetitions, passing to the third step 77 polyline pairs. Executing this last step we obtained 22 intersections and 55 non-intersections. Thus, we can observe that the filter's use reduced in 50% (76/153) the number of comparisons to be done in the last step and that 12% (3/25) of the intersections and 57% (73/128) of the non-intersections were detected.

In order to demonstrate the filter's efficiency, tests were done executing the architecture's steps alternately and timing each execution. The computer used was a Pentium 166Mhz with 32Mb of memory. The results are presented in table 3.

Steps	1, 2 and 3	1 and 2	1	1 and 3
Elapsed Time (s)	1,48	0,11	0,05	2,58

Table 3 –Results obtained in successive executions

Analyzing table 3 one can observe that the use of the filter reduces in 43% ((2,58-1,48)/2,58) the time spent when compared with the execution without the second step. Besides this, we confirmed that the execution of the first and second steps are insignificant compared to the time spent in the execution of the third step, as pointed in [BRIN94]. In the execution of all three steps about 3,4% (0,05/1,48) of the total time is spent in the first step, 4% ((0,11-0,05)/1,48) in the second, while the third step spends 92,6%, as presented in the table 4.

Step	1	2	3	Total
Elapsed Time (s)	0,05	0,06	1,37	1,48
%	3,4	4	92,6	100

Table 4 – Elapsed time per step when executing all three steps

Executing the architecture for a join of part of New York's highways and rivers, the first step result in 4784 MBR comparisons of which 1124 overlapped, considering polyline repetition

as previously presented, advancing to the following step. The second step resulted in 10 intersections and 768 non-intersections, eliminating repetitions, leaving to the third step 346 polyline pairs. In the execution of this last one we obtained 58 intersections and 288 non-intersections. Thus, we can observe that the filter's use reduced in 69% (778/1124) the number of comparisons to be done in the last step and that 15% (10/68) of the intersections and 73% (768/1056) of the non-intersections were detected. Even better results than the previous test were obtained since the percentage of conclusions increased for intersections and for non-intersections.

The time spent was also observed as shown in table 5.

Steps	1, 2 and 3	1 and 2	1	1 and 3
Elapsed Time (s)	3,79	0,55	0,28	11,2

Table 5 –Results obtained in successive executions

Table 5 demonstrates the efficiency of the 5CDRS filter which reduces in 66% ((11,2-3,79)/11,2) the time spent when compared with the execution without the second step. Besides this, we noted that the execution of the first and second steps remain insignificant comparing to the time spent in the execution of the third step. Executing all three steps about 2,5% (0,28/11,2) of the total time is spent in the first step, 2,4% ((0,55-0,28)/11,2) in the second, while the third step spends 95,1%, as presented in the table 6.

Step	1	2	3	Total
Elapsed Time (s)	0,28	0,27	10,65	11,2
%	2,5	2,4	95,1	100

Table 6 – Elapsed time per step when executing all three steps

6. Conclusions

In this work, we presented the use of the 5CDRS approximation, specifically designed for performing spatial joins of polyline sets, fitting in the MSQP's second step. According to the results obtained in [BRIN93] and [ZIMB98] we can state that improvements in step 2 reduces the size of the input to step 3, reducing significantly the join's total cost. Our approach, in fact, did reduce the number of elements left to step 3 by about 50% when compared with the standard approach, as already stated in [ZSMA00], which is just the MBR join. The decision capacity of our filter is similar to filters proposed in [BRIN93], which are 46% in the average. Just as an example, in [BRIN93], in the join without filters using MBR R-Trees, the *plane-sweep* algorithm consumes about 80% of the join total time. When the filters are applied, the total time is reduced in about 40%; fact confirmed in the tests described in the previous section, whose time reduction in the third step was superior to 50%.

As future work, we have the evaluation of the use of our approach in the performance of joins between sets of polygons and polylines.

7. References

[ARG98] Arge, L., et al. 1998. "Scalable Sweeping Based Spatial Join". In: 24th International Conference on Very Large Databases, New York City, NY, USA, August 1998.

[BERC96] S. Berchtold, D. A. Keim, H. P. Kriegel: “The X-Tree: An Index Structure for High-Dimensional Data”, In Proceedings of 22nd International Conference on Very Large DataBases, Bombay, India, 1996.

[BRIN93] T. Brinkhoff, H. P. Kriegel, and B. Seeger: “Efficient processing of Spatial Joins Using R-Trees”. In Proceedings of the 1993 ACM-SIGMOD Conference, Washington, DC, USA, May 1993.

[BRIN93b] T. Brinkhoff, H. P. Kriegel, and R. Schneider: “Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems”. In Proceedings of 9th International Conference on Data Engineering, Vienna, Austria, 1993.

[BRIN94] T. Brinkhoff, H. P. Kriegel, R. Schneider, and B. Seeger: “Multi-step Processing of Spatial Joins”. In Proceedings of the 1994 ACM-SIGMOD Conference, Minneapolis, USA, May 1994.

[BRIN95] T. Brinkhoff, H. P. Kriegel, R. Schneider, and A. Braun: “Measuring the Complexity of Polygonal Objects”. In Proceedings of ACM International Workshop on Advances in Geographic Information Systems, Baltimore, MD, USA, December 1995.

[BRIN96] T. Brinkhoff, H. P. Kriegel And B. Seeger: “Parallel Processing of Spatial Joins Using R-trees”, In Proceedings of 12th International Conference on Data Engineering, New Orleans, LA, USA, 1996.

[DEWI96] D. DeWitt and J. M. Patel: “Partition Based Spatial-Merge Join”. In Proceedings of the 1996 ACM-SIGMOD Conference, Montreal, Canada, June 1996.

[ESPE97] Esperança, C., Samet, H. 1997. “Orthogonal Polygons as Bounding Structures in Filter-Refine Query Processing Strategies”. In: Proceedings of the 5th International Symposium on Spatial Databases, Berlin, Germany, June 1997.

[HUAN97] Huang, Y. W., Jones, M. C., Rundensteiner, E. A. 1997a. “Improving Spatial Intersect Joins Using Symbolic Intersect Detection”. In: Proceedings of the 5th International Symposium on Advances in Spatial Databases, SSD'97, Berlin, Germany, July 1997.

[KOUD97] Koudas, N., Sevcik, K. 1997. “Size Separation Spatial Join”. In: Proceedings of ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, USA, May 1997.

[MAMO99] Mamoulis, N., Papadias, D. 1999. “Integration of Spatial Join Algorithms for Joining Multiple Inputs”. In: Proceedings of the ACM Conference on the Management of Data (SIGMOD), Philadelphia, PA, USA, May 1999.

[MING95] M. L. Lo and C. V. Ravishankar: “Generating Seede Trees From Data Sets”. In Proceedings of 4th International Symposium on Large Spatial Databases, Portland, ME, USA, August 1995.

[MING96] M. L. Lo and C. V. Ravishankar: “Spatial Hash-Joins”. In Proceedings of the 1996 ACM-SIGMOD Conference, Montreal, Canada, June 1996.

[SAIF94] MELP Spatial Archive and Interchange Format (SAIF): Formal Definition Release 3.1 April 1994 Reference Series Volume 1 - Surveys and Resource Mapping Branch - Ministry of Environment, Lands and Parks (MELP) - Province of British Columbia - Canada.

[SAME90] H. Samet: “The Design and Analysis of Spatial Data Structures”, Addison-Wesley Publishing Company, 1990.

[SQL395] X3H2-95-084/DBL:YOW-004, (ISO-ANSI Working Draft) Database Language SQL (SQL3),Jim Melton (ed.), March, 1995

[VEEN95] H. M. Veenhof, Peter M. G. Apers, Maurice A. W. Houtsma: “Optimization of Spatial Joins Using Filters”. In Advances in Databases, 13th British National Conference on Databases, BNCOD 13, Manchester, United Kingdom, July 1995.

[ZIMB97] Zimbrão, G., Souza, J. M. 1997b. “Realização Eficiente de Junções Espaciais Utilizando Hash-Join”. In: Anais do XII Simpósio Brasileiro de Banco de Dados - Fortaleza, CE, Brasil, Outubro, 1997.

[ZIMB98] Zimbrão, G., Souza, J. M. 1998. “A Raster Approximation For Processing of Spatial Joins”. In: Proceedings of VLDB’98 - 24th International Conference on Very Large Databases, New York City, NY, USA, Aug 1998.

[ZSMA00] Zimbrão, G., Souza, J. M., Monteiro, R. S., Azevedo, L. G., 2000. “Filtro Raster para Junção de Polilinhas”. In: Anais do XV Simpósio Brasileiro de Banco de Dados - João Pessoa, PA, Brasil, Outubro, 2000.