Multi-Step Processing of Spatial Joins

Thomas Brinkhoff
Hans-Peter Kriegel
Ralf Schneider
Bernhard Seeger

Institute for Computer Science, University of Munich
Leopoldstr. 11 B, D-80802 München, Germany

e-mail: {brink,kriegel,ralf,bseeger}@informatik.uni-muenchen.de

Abstract

Spatial joins are one of the most important operations for combining spatial objects of several relations. In this paper, spatial join processing is studied in detail for extended spatial objects in two-dimensional data space. We present an approach for spatial join processing that is based on three steps. First, a spatial join is performed on the minimum bounding rectangles of the objects returning a set of candidates. Various approaches for accelerating this step of join processing have been examined at the last year's conference [BKS 93a]. In this paper, we focus on the problem how to compute the answers from the set of candidates which is handled by the following two steps. First of all, sophisticated approximations are used to identify answers as well as to filter out false hits from the set of candidates. For this purpose, we investigate various types of conservative and progressive approximations. In the last step, the exact geometry of the remaining candidates has to be tested against the join predicate. The time required for computing spatial join predicates can essentially be reduced when objects are adequately organized in main memory. In our approach, objects are first decomposed into simple components which are exclusively organized by a main-memory resident spatial data structure. Overall, we present a complete approach of spatial join processing on complex spatial objects. The performance of the individual steps of our approach is evaluated with data sets from real cartographic applications. The results show that our approach reduces the total execution time of the spatial join by factors.

1 Introduction

Recently, spatial database systems have become more and more important for public administration, science and business. Several *spatial database systems (spatial DBSs)*, particularly designed for organizing spatial data of a geographic information system (GIS), have been developed for applications such as cartography, environmental science and geography. For these applications, the data volume is extremely high, the spatial objects show a very complex structure and the computation of spatial operators is time-intensive. Therefore, the requirements on a spatial DBS are particularly related to efficient query processing.

A spatial object consists of (at least) one spatial attribute that describes the geometry of the object. This attribute contains two- or three-dimensional data of a common type such as points, lines, rectangles, polygons, surfaces and even more complex types composed from simple types. In analogy to relational DBSs, a collection of spatial objects defined on the same attributes is called a *spatial relation*. For example, the spatial relation *Cities (CName, Postal Code, Population, CRegion)* contains the geometric attribute *CRegion* that describes the border of the city using a two-dimensional polygon. A spatial query is related to the geometric attribute of the spatial object. A typical spatial query is the window query where the response set consists of all objects whose geometric component overlaps with a given rectilinear query rectangle.

In contrast to a window query, the *spatial join* is defined on two or more relations. The spatial join computes a subset of the Cartesian product. It combines spatial objects from these relations according to their geometric attributes, i.e. these attributes have to fulfill a spatial predicate. For example, consider the spatial relation *Forests (Id, Name, FRegion)* where the geometric attribute *FRegion* represents the borders of forests. The query "find all forests which are in a city" is an example of a spatial join on the relations *Forests* and *Cities*. Here, the spatial predicate is whether a forest intersects a city. Since spatial joins frequently occur in a spatial DBS, their importance is comparable to the one of traditional joins in a relational DBS.

In this paper, we investigate spatial join processing for two sets of two-dimensional spatial objects whose geometry corresponds to polygonal areas. The geometry is assumed to be in vector representation, i.e. it is given as a sequence of two-dimensional points. We assume that the corresponding spatial operator tests polygonal areas for intersection. For other predicates, e.g. inclusion, a similar approach can be used as presented for intersection. In particular, our emphasis is put on non-simple objects whose description consists of hundreds of points. Then, for the spatial join, some of the most expensive operations are to transfer large spatial objects from disk into main memory and to compute the join predicate of spatial objects. In order to improve the performance of these operations, we propose to process a spatial join in three steps. Each step uses a different representation of the objects.

- First, instead of using an exact representation, the minimum bounding rectilinear rectangle is used as an approximation for computing an approximate spatial join. This step returns a so-called candidate set that contains all of the answers and additionally elements of the Cartesian product which do not fulfill the join predicate. We have demonstrated that this step can be efficiently supported by spatial access methods such as R*-trees [BKS 93a].
- In the second step, more accurate approximations are exploited for filtering out elements (false hits) from the candidate set that do not fulfill the join predicate. Moreover, answers can already be identified using conservative as well as progressive approximations without accessing the exact representation of the spatial objects.

- Eventually, in the third step, all remaining members of the candidate set are examined for being answers to the spatial join. This step requires access to the exact representation of the spatial objects. The time required for checking objects against the join predicate can be essentially reduced by using an appropriate representation of the objects in main memory.

Although join processing has been studied in the literature extensively, see [ME 92] for a survey, the spatial join has attracted much less attention. Because almost all approaches for performing traditional joins cannot efficiently be applied to spatial joins, previous approaches are based on new join algorithms that exploit spatial access methods. Most previous work on spatial joins was done under the assumption that spatial objects are simple. Orenstein [Ore 86] has proposed spatial join processing for objects that can be constructed as a union of cells in a two-dimensional grid. Our discussion of spatial joins [BKS 93a] was restricted to rectilinear rectangles. This work yields the basic building block for the first of the above mentioned three steps. Günther [Gün 93] has presented a general model for estimating the cost of spatial joins. The work of Blankenagel and Güting [BG 90] is an exception because they provide an algorithm for a spatial join on two sets, where the one contains polygonal areas and the other consists of two-dimensional points. Their approach is based on techniques from the area of computational geometry.

Algorithms for supporting spatial joins on simple objects can be used as an implementation for the first step of the spatial join on more complex objects. However, the authors are not aware of a complete approach for spatial join processing including the computation of the response set from the candidate set. In particular, this computation may determine the response time of spatial joins. In this paper, we address the problem how to improve the computation of the response set.

The paper is organized as follows. In section 2, we motivate and introduce multi-step processing and present the basic steps of the spatial join processor. In section 3, several approximations are introduced for performing the filter steps of a spatial join. The part of the query processor dealing with the exact geometry is presented in section 4. The total performance improvements are discussed in section 5. Section 6 concludes the paper and gives an outlook to future work.

2 Basic Ideas of Spatial Query Processing

Due to extremely large data volumes and also due to the high complexity of objects and queries, spatial DBSs impose stringent requirements on their query processing. *Spatial queries*, such as *point queries, window queries*, *nearest neighbor queries*, and *spatial joins*, are the basic operations in a spatial DBS. In particular, they serve as building blocks for more complex and application-defined operations, e.g. for the map overlay in a geographic information system (GIS). Therefore, an efficient implementation of spatial queries is an important requirement for good overall performance of a spatial DBS and of applications based on it.

In this section, the basic ideas of our approach are presented. First, we briefly introduce spatial objects and spatial joins. Then, a first approach for spatial join processing is discussed and its most expensive operations are identified. Eventually, we outline the architecture of our spatial query processor that is based on multi-step query processing.

2.1 The geometry of spatial objects

In the following, the geometry of a spatial object is assumed to be described by a sequence of two-dimensional points. It is obvious that this representation allows to model points, line segments and polygons directly. In this paper, our emphasis is put on extended spatial objects, especially on *polygons* and *polygons with holes*. A *polygon with holes* consists of an outer polygon and an arbitrary number of "internal" polygons cut out from the outer polygon. For example, the geometry of the spatial objects from the relation *Forest* can be well described using polygons with holes. The "holes" might represent areas such as lakes.

Spatial objects exhibit a high complexity with respect to the following parameters [BHKS 93]: 1.) their number of points (vertices), 2.) their extension, 3.) their shape, and 4.) their distribution in data space. As a consequence of the complexity of the objects, geometric operations, e.g. the test whether two objects intersect, become very expensive.

2.2 Spatial join

The spatial join is one of the most frequently used operations in a spatial DBS. The performance of a spatial DBS is likely to be affected more by a spatial join rather than by window queries because a spatial join has to access objects several times. Its execution time is generally superlinear in the number of objects.

A *relational* θ*-join* of two relations *A* and *B* on columns *i* and *j*, denoted by $A \underset{i\theta j}{\bowtie} B$, combines those tuples where the *i*-th column of *A* and the *j*-th column of *B* fulfill the predicate θ. The most important join is the *equijoin* where θ is equality. A join $A \underset{i\theta j}{\bowtie} B$ is called *spatial join* if the *i*-th column of *A* and the *j*-th column of *B* are spatial attributes and if θ is a spatial predicate [Gün 93]. Hence, the spatial join computes a subset of the Cartesian product of relations *A* and *B*. For example, instances of spatial attributes can be line segments representing rivers, railway tracks and highways or polygons representing a part of the surface of the earth. Spatial predicates may be intersection, containment or other proximity predicates.

The most important spatial join is the *intersection join* where θ is intersection. In this paper, our discussion is restricted to the intersection join. However, many of the results can easily be transferred to spatial joins using other spatial predicates. For an intersection join, non-spatial attributes are not relevant and therefore, they are not considered anymore in the remainder of the paper. The relations are given as two sets $A = \{a_1,\ldots,a_n\}$ and $B = \{b_1,\ldots,b_m\}$ of spatial objects where *n* and *m* refer to the number of objects in relation *A* and *B*, respectively.

2.3 A first approach for spatial join processing

Despite of the importance of spatial joins, most research efforts have so far focussed on the investigation of natural joins and equijoins. Unfortunately, almost all methods designed for processing relational θ*-joins* cannot be used for spatial joins without modifications. Hashed-based joins are not appropriate for performing spatial joins since spatial objects which fulfill the join predicate are not in a common hash bucket in general. Sort-merge joins can be applied to spatial joins, but this requires a one-dimensional ordering of the two-dimensional objects. Orenstein [Ore 86] has proposed to sort objects using a space-filling curve. However, his approach was restricted to simple objects and hence, it can only be considered for computing a candidate set, i.e. a superset of the response set. Only the approach of nested loops can be used without any modifications for both relational and spatial joins. Nested loops will serve as an initial starting point. The nested loops approach is given as follows:

```
ALGORITHM Nested-Loops;
  RS := ∅;                   (* initialize the response set *)
  FOR i := 1 TO n DO         (* outer loop *)
    obj_A := Fetch(A,i);     (* reading the i-th object of A *)
    FOR j := 1 TO m DO       (* inner loop *)
      obj_B := Fetch(B,j);   (* reading the j-th object of B *)
      IF obj_A ∩ obj_B ≠ ∅ THEN   (* test the join predicate *)
        RS := RS ∪ {(obj_A, obj_B)};
                             (* insert pair in the response set *)
END Nested-Loops;
```

This simple algorithm illustrates the most expensive operations of spatial join processing. First, the transfer of an object from disk into main memory can be a very expensive operation when the object is large. In particular, large spatial objects might prevent keeping the relation *B* resident in main memory, although *m* (the number of elements in *B*) may be rather small. In the worst case, an element of *B* has to be read *n* times. Second, the test of the join predicate can be very expensive. In particular, this can become the bottleneck if the description of the objects consists of a large number of points. Third, as it is true for all nested loops, the number of loops is very high. In the following subsection, we briefly outline the basic idea of our approach for reducing the cost of spatial join processing.

2.4 Multi-step processing of spatial joins

Our goal is the design of a multi-step query processor that improves the performance of spatial join processing. In former publications ([KBS 93], [BHKS 93]), we presented a *spatial query processor* for spatial queries such as the point and the window query. This spatial query processor is based on a multi-step-procedure. Its main goal is to accelerate expensive steps by preceding filter steps which reduce the number of spatial objects investigated in an expensive step. This multi-step-procedure can be extended to spatial join processing.

A spatial join is abstractly executed as a sequence of steps. First, spatial access methods (SAMs) are used for restricting the search space. SAMs traditionally organize *minimum bounding rectangles (MBRs)* as geometric keys. Thus, a SAM is not able to yield the exact result of the join, but a *set of candidate pairs* that contains all answers and additionally pairs of objects that do not fulfill the join predicate. In the next step, these candidate pairs are examined using a *geometric filter*. The geometric filter investigates accurate approximations of the objects in the set of candidates, i.e. pairs of approximations are tested for the join predicate. As a result, three classes of answers are identified: *hits* fulfilling the join predicate, *false hits* not fulfilling the join predicate and remaining *pairs of candidates* possibly fulfilling the join predicate. The third step operates on the exact geometry of the objects. The remaining candidate pairs have to be investigated whether they fulfill the join predicate or not. This expensive step is handled by the exact geometry processor and can be improved by using efficient computational geometry algorithms. This multi-step approach to spatial join processing is illustrated in figure 1.

As described above, the first step computes the spatial join for the minimum bounding rectangles (MBRs) of the objects in relations *A* and *B*. This operation is called the *MBR-join*. The execution time of the MBR-join is less than the one of the original join for the following reasons. First, instead of fetching the exact representation of a complex spatial object, only its MBR has to be read from disk. Second, testing the MBRs against the join predicate is much cheaper than testing the exact objects. However, due to the high number of loops, the nested loops approach is still very inefficient for processing MBR-joins. For efficiently scaling down the search space, spatial access methods are used which are readily available in most spatial DBSs. We showed that the *R*-tree* [BKSS 90] - one of the most efficient members of the *R-tree* family [Gut 84] - can also be exploited for performing spatial joins efficiently [BKS 93a]. In an experimental performance comparison using real cartographic data, the number of page accesses required for performing the spatial join was shown to be close to optimal. Moreover, due to the sophisticated techniques of restricting the search space and due to spatial sorting, the number of tests of MBRs against the join predicate could be kept very low, see [BKS 93a]. Obviously, instead of R*-trees, any other spatial access methods such as R^+-trees [SRF 87] or approaches based on space filling curves [Fal 88, Jag 90b] might be considered for implementing the MBR-join.

As a result of the MBR-join, we obtain a set of candidate pairs, candidate set for short. The goal of the second step (geometric filter) is to identify hits as well as false hits in the candidate set without investigating the exact representation of the objects. For spatial joins, several approaches are discussed and compared in full detail in section 3. The third step requires processing the exact geometry of the remaining objects in the candidate set. In order to perform the test against the join predicate efficiently, appropriate representations of spatial objects in main memory are presented and discussed in section 4.

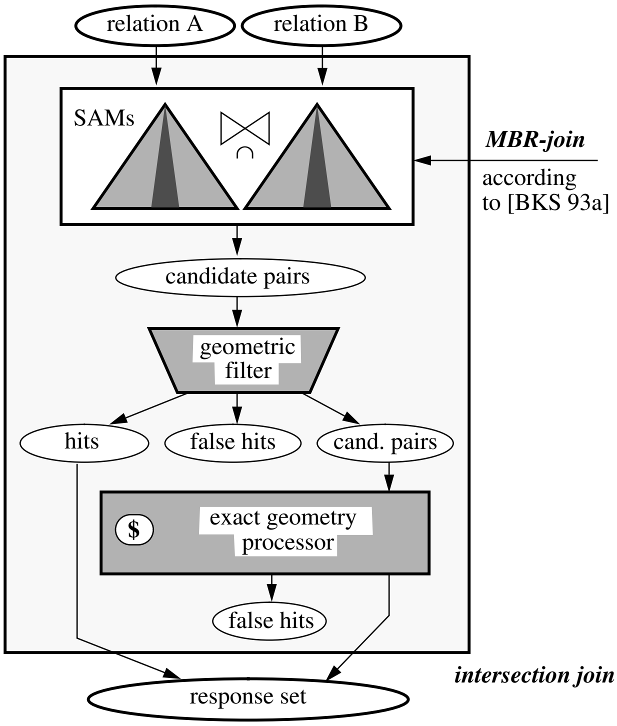


Figure 1: Spatial join processor

Note that the sets of candidates are *not* stored as intermediate results. Instead, a computed pair of candidates (or a small set of them) is immediately given as input to the subsequent step. In consequence, no additional cost arises for handling these candidates.

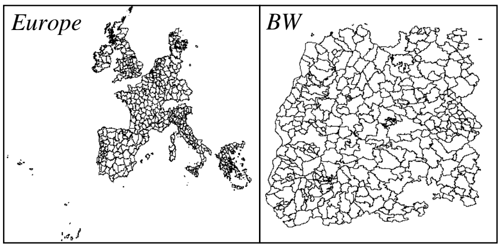
3 Exploiting Approximations for Spatial Joins

A spatial access method organizes the spatial objects according to a geometric key. The minimum bounding rectangle (MBR) is the most popular geometric key. Using the MBR, the complexity of an object is reduced to four parameters where the most important features of the object (position and extension) are maintained. A further important advantage of the MBR is the fast execution of spatial operations like the point-in-MBR test or the test for intersection. Consequently, the MBR is widely used. Testing the MBRs of the objects against the spatial query condition, we obtain candidates possibly fulfilling the query (see figure 1). This set of candidates is the starting point for the following investigations.

Using MBRs provides a fast but inaccurate filter for the set of answers. The larger the area of the MBR differs from the area of the original object, the more inaccurate is the geometric filtering, i.e. the candidate set includes a lot of false hits. In order to quantify this statement, we experimentally investigate the quality of MBR-approximations using real cartography data.

3.1 Evaluation of MBR-approximation

In order to get expressive and realistic results on the suitability of the MBR-approximation for supporting point queries (not for join processing), we investigated various real spatial data sets [BKS 93b]. In the following, we restrict our experiments to two representative spatial relations with different resolutions. The first relation *Europe* depicts the counties of the European Community in 1989. The second relation is *BW* representing some municipalities of Baden-Württemberg, a state in Germany. Figure 2 depicts the analyzed spatial relations and lists some characteristics. $m_{\varnothing}$, $m_{\min}$ and m_{max} denote the average, the minimum and the maximum number of vertices occurring in the relation, respectively.



relation	# objects	$m_{\varnothing}$	m_{min}	m_{max}
Europe	810	84	4	869
BW	374	527	6	2087

Figure 2: The analysed spatial relations

In the literature, several alternatives are proposed to evaluate approximations. In our application, we are interested in measuring the accuracy of the geometric filter. For polygonal objects, the accuracy of the filter step is increased by decreasing the deviation of the approximation from the original object. In the following, we call the difference between the area of an object and the area of its approximation *false area*. For comparability, we normalize the false area to the area of the approximated object *(normalized false area)*. Table 1 shows impressively that real cartography objects are only roughly approximated by MBRs. ∅ is the average of the normalized false areas; *min* and *max* denote the minimum and maximum values occurring in the spatial relation, respectively.

	∅	*min*	*max*
Europe	0.91	0.25	20.13
BW	1.02	0.38	3.48

Table 1: False area of the MBR normalized to the object area

Performing a point query, the fraction of false hits is proportional to the normalized false area of the MBR. In other words, the higher the normalized false area of the MBR is, the more often the exact object representation is unnecessarily loaded into main memory and tested with costly computational geometry algorithms.

Contrarily to point queries, we cannot postulate a direct dependency of normalized false area and number of false hits for spatial joins because spatial objects are tested against each other in pairs. Therefore, the ratio of candidates, false hits and hits is determined by empirical tests with real cartography data. As basic data to investigate the spatial join, we use the spatial relations from figure 2. In order to obtain suitable test series, we pursue two strategies: Performing strategy *A*, the original relation (*Europe* and *BW*, respectively) is taken as first relation of the join. The second relation is generated by shifting the objects of the first relation in x- and y-direction. The test series generated following strategy *A* are called *Europe A* and *BW A*. The other strategy *B* generates two spatial relations per test series randomly shifting and rotating the objects of the original relation. Additionally, the polygons are scaled in such a way that the sum of the object areas is equal to the area of the data space. Contrary to strategy *A*, an object of a relation generated by strategy *B* may intersect other objects of the same relation. The test series generated following strategy *B* are called *Europe B* and *BW B*. Table 2 depicts the characteristics of the four test series.

test series	# intersecting MBRs	# hits	# false hits
Europe A	1858	1273	585
Europe B	4816	3203	1613
BW A	2253	1504	749
BW B	2562	1684	878

Table 2: Test data for approximation joins

Table 2 demonstrates that about one third of the object pairs computed by the MBR-join are false hits, i.e. although the MBRs of the objects intersect, the objects themselves do not intersect. The reason for the high number of false hits is the rough approximation of MBRs which leads to high values for the normalized false area. Consequently, there is a great potential to improve the processing of spatial joins by using approximations in the geometric filter which approximate the spatial objects more exactly than the MBR. In the next subsection, we present a detailed investigation of such approximations for intersection joins.

3.2 Reducing the number of false hits

In order to identify more false hits, we examine approximations which describe the objects better than the MBR. Approximations of objects which are used for geometric filtering should be simple to provide a fast filter (*simplicity criterion*) and they should have a good accuracy (*quality criterion*).

The MBR is a conservative approximation. An approximation is called *conservative* iff each point inside the contour of the original object is also in the conservative approximation. The basic idea is to perform the test for the query condition on conservative approximations with a better accuracy than the MBR. This is expected to be much cheaper than performing the tests on the exact objects. If the test fails, the candidate is a false hit. Otherwise, the candidate remains in the candidate set and it has to be investigated in the following steps. A detailed investigation of this effect for performing point queries has been presented in [BKS 93b].

Conservative approximations can be grouped into *convex* and *concave approximations*. Due to the simplicity criterion, we restrict our considerations to the class of convex approximations. First, computational geometry algorithms for concave polygons are considerably more complicated and time intensive than for convex polygons. Second, efficient algorithms for computing a conservative approximation are well-researched.

Approximations

In order to improve the processing of spatial joins, we investigate five convex conservative approximations. Let n denotes the number of vertices of a spatial object.

Rotated minimum bounding rectangle (RMBR): The rotated MBR (RMBR for short) is an MBR where additionally a rotation is allowed. It is computed by a simple $O(n^2)$-algorithm. Five parameters are necessary to store the RMBR.

Convex hull (CH): An obvious and popular approximation for simple polygons is the convex hull. The construction of the convex hull is one of the best understood problems in computational geometry. Graham's simple scan-algorithm [PS 85] requires $O(n \log n)$ time for the computation of a convex hull. The storage requirement for the convex hull is determined by the form and complexity of the object geometry; it varies from object to object.

Minimum bounding m-corner (m-C): To obtain a predefined constant storage requirement, it is possible to compute the minimum bounding m-corner starting from the convex hull of the polygon. For the first time, an algorithm to construct the m-C-approximation was proposed by Dori and Ben-Bassat [DB 83] which has a time complexity of $O(n \log n)$.

Minimum bounding circle (MBC): The circle needs three describing parameters for specifying the center of the circle and the radius. In our tests, we used a randomized algorithm with an expected linear complexity [Wel 91].

Minimum bounding ellipse (MBE): Two-dimensional ellipses are determined by 5 parameters. We computed the MBE by a randomized algorithm [Wel 91] which has an expected linear complexity.

Figure 3 visualizes the selected approximations using Great Britain as an example. These approximations differ especially in their accuracy and number of parameters which is given in brackets. The convex hull has the highest storage requirement (and the best accuracy) and the circle the lowest storage requirement.

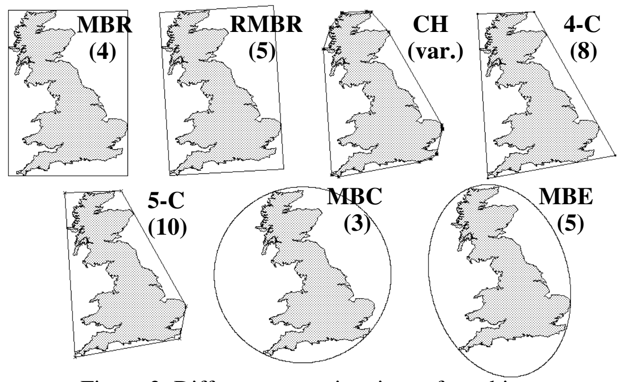


Figure 3: Different approximations of an object

Approximation quality of the different approximations

For defining a quality measure for approximations, we have to consider the false area. If the approximation is stored *additionally to* the MBR, we use the *MBR-based false area* as a measure for the *approximation quality*. That means, we first compute the intersection *I* of the approximation and the MBR and then the false area between the object and the intersection *I*. This is necessary because the MBR is tested first and the approximation may cover parts of the data space which are not covered by the MBR. In figure 4, the average approximation quality is presented for different approximations and the spatial relations *BW* and *Europe*. For comparability, the MBR-based false areas are normalized to the object areas.

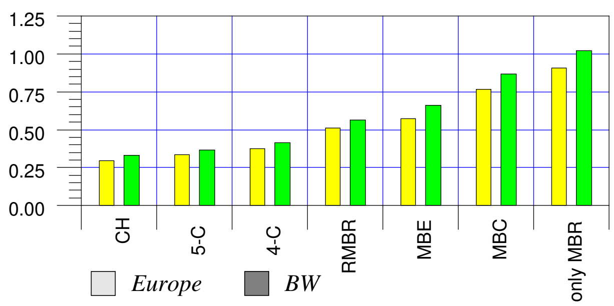


Figure 4: MBR-based false area normalized to the object area

The results show that the more parameters are available for the representation of an additional approximation, the better is its approximation quality. The area of the 5-corner is nearly as accurate as the area of the convex hull. The storage requirements of convex hulls vary extremely, in our comparisons between 4 and 80 parameters. The convex hull needs on the average 26 parameters for *Europe* and 46 for *BW* whereas the 5-corner requires only 10 parameters. The MBR-based false area of a RMBR is also considerably less than the one of the MBR requiring only one additional parameter.

As mentioned before, if conservative approximations of two objects do not intersect, it follows that also the objects do not intersect. For spatial joins we expect - in correspondence to the point query - to identify considerably more false hits by additional approximations. This expectation is confirmed by our test results depicted in table 3.

test series	Percentage of identified false hits using					
	MBC	MBE	RMBR	4-C	5-C	CH
Europe A	17.9	42.1	35.7	50.9	66.3	80.7
Europe B	19.2	44.0	45.2	58.6	69.1	82.8
BW A	17.6	43.7	45.3	59.1	70.2	82.1
BW B	16.2	44.1	37.2	52.4	64.7	79.7

Table 3: False hits identified by approximations

The columns give the percentage of identified false hits when the specified approximation is used in the geometric filter after first performing the MBR-join. The results demonstrate that a high percentage of false hits can be identified by using these approximations. For example, the 5-C detects 68% of the false hits. Thus, only 32% of the non-intersecting objects whose MBRs intersect, must be transferred into main memory and investigated in the exact geometry test.

In figure 5, we depict the dependency between MBR-based false area and identified false hits for the *Europe B* test series and for various approximations. Considering in figure 5 the MBR, the MBC, the RMBR, the 4-C, and the object itself, we recognize that the dependency is almost linear. However, the deviation of the 5-C, MBE, and the convex hull demonstrates that not only the false area, but also the adaptability to the object is an important property of an approximation for identifying non-intersecting objects.

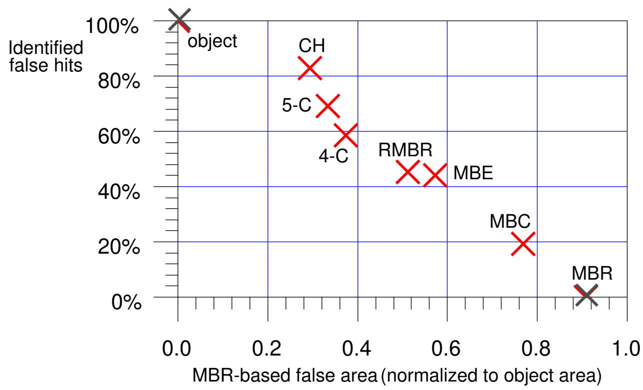


Figure 5: Dependency between MBR-based false area and percentage of identified false hits

Because of its good approximation quality, the convex hull shows the best results. But its number of vertices varies extremely and its average storage requirements are by factors higher than the requirements of the other approximations. Therefore, convex hulls are not suitable to be stored in the data pages of a spatial access method which needs high numbers of entries within its data pages in order to achieve a good query performance. Overall, the 5-corner is a good compromise: it needs much less storage, but it has a very good approximation quality and detects about two thirds of the non-intersecting object pairs delivered from the MBR-join. The investigation in section 3.4 demonstrates the gains obtained by using the RMBR and the 5-C.

3.3 Identifying hits in geometric filtering

In the previous subsection, we strived for reducing the number of false hits. As depicted in table 2, the number of hits clearly exceeds the number of false hits. After avoiding a large percentage of false hits, the ratio false hits to hits approaches 1 to 5. Therefore, it is not reasonable to invest more time in identifying further false hits. Instead, we examine techniques which identify intersecting objects *(hits)* without inspecting the exact geometry. In this subsection, two techniques are presented: 1.) the false-area test and 2.) progressive approximations.

The false-area test

We call the difference between the area of an object *obj* and the area of its conservative approximation the *false area* (denoted by $fa_{Appr}(obj)$). For two intersecting polygonal objects obj_1 and obj_2 the following property holds:

$$Appr(obj_1) \cap Appr(obj_2) \; > \; fa_{Appr}(obj_1) + fa_{Appr}(obj_2) \;\Rightarrow\; obj_1 \cap obj_2 \neq \emptyset$$

To say it in words, if the area of the intersection of the approximations is larger than the sum of the false areas of the objects, it follows that the objects intersect. This property can be exploited for processing spatial joins if the false area is stored additionally to the approximation for each object. This requires only one additional parameter. Now, the most interesting question is how many hits can be identified by the false-area test. Table 4 gives the percentage of identified hits in our experiments for various approximations.

Due to its bad approximation quality, the false-area test does not pay off for the MBR: almost no hit can be identified. Performing the test with the 5-corner - our favorite approximation from the last subsection - identifies about 6% of the hits. These results motivate to look for a test which identifies more hits than the false-area test. For such a test, we allow a higher number of parameters.

test series	Percentage of identified hits using the false-area test with				
	MBR	RMBR	4-C	5-C	CH
Europe A	0.1	0.4	3.8	8.1	12.5
Europe B	0.1	0.3	1.9	5.2	8.8
BW A	0	0.9	2.6	6.0	10.3
BW B	0	0.3	1.7	5.3	8.8

Table 4: Hits identified by false-area test

Progressive approximations

In addition to conservative approximations which were discussed before, we will now consider another type of approximations. For identifying hits *progressive approximations* are adequate. A polygonal object is progressively approximated if the point set of the approximation is a subset of the point set of the object. Figure 6 depicts examples of conservative and progressive approximations. When the low-cost test for progressive approximations is successful, we obtain a definite answer; only a failed operation triggers the costlier operation on the exact object description. For the intersection join this implies that if two progressive approximations intersect, it follows that the objects intersect.

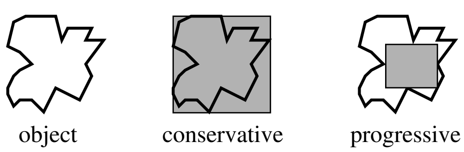


Figure 6: Conservative and progressive approximations

It may be intuitively clear that it is more expensive to compute progressive approximations than conservative ones. This holds especially true if a maximum enclosed approximation is computed. Therefore, we investigate the two simplest progressive approximations: the *maximum enclosed circle* and the rectilinear *maximum enclosed rectangle*.

Maximum enclosed circle (MEC): It can be computed for a simple polygon by using the voronoi diagram of its edges. One of the nodes of the diagram is the center of the MEC.

Maximum enclosed rectangle (MER): In order to simplify the computation of an enclosed rectangle, we restricted the investigation to rectangles which fulfill the following two properties: 1.) they intersect the longest enclosed horizontal connection starting in a vertex of the polygon and 2.) the x- and the y-coordinates of the rectangles are x- and y-coordinates of vertices of the polygon. This type of enclosed rectangles is called maximum enclosed rectangle in the remainder of the paper.

Figure 7 depicts a Bavarian region (Oberpfalz) approximated by an MEC and an MER.

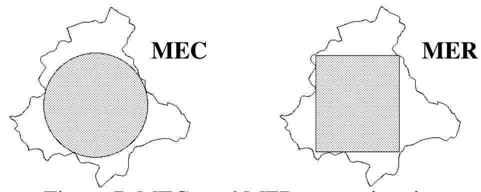


Figure 7: MEC- and MER-approximations

For evaluating the accuracy of progressive approximations, we use as a measure the area of the approximation normalized to the area of the corresponding polygon. In order to get a flavor of both presented approximations for real geographic data, we computed the normalized approximation areas for the relations *Europe* and *BW*. Figure 8 depicts the results. The experiments show that on the average 42% of the area of a polygon is covered by the enclosed circle and 44% by the enclosed rectangle. With respect to the three and four parameters for representing the MEC and MER, respectively, these values are pleasantly high.

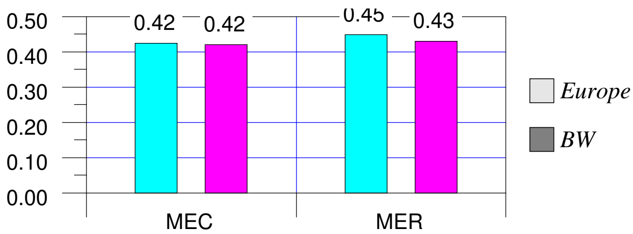


Figure 8: Approximation quality of the progressive approximations

As mentioned before, if the progressive approximations of two objects intersect, it follows that the objects intersect, too. Considering the processing of spatial joins, the main question is for the number of identified hits. Table 5 shows the results of our tests. The results in the column ‘MEC’ demonstrate that almost 32% of the hits are identified by the MEC-approximation without accessing the exact geometry. The MER-approximation even allows to find around 35% of the hits. These percentages are considerably higher than those gained by the false-area test which identifies only about 6% using the 5-corner. Storing three or four additional parameters instead of one, the number of identified hits increases by a factor of 5 to 6. Because of the higher number of identified hits, the MER seems to be the more suitable progressive approximation, although it needs one parameter more than the MEC.

Further experiments demonstrate that a combination of progressive approximations and the false-area test does practically not improve the rate of identified hits. Therefore, it is not reasonable to use the false-area test in addition to progressive approximations.

test series	Percentage of identified hits using	
	MEC	MER
Europe A	31.4	36.2
Europe B	31.8	35.3
BW A	31.6	34.3
BW B	32.6	33.6

Table 5: Identified hits by progressive approximations

3.4 The organization of approximations in SAMs

The last sections have demonstrated the potentials induced by using conservative approximations with a high approximation quality and by using progressive approximations. In this section, we discuss the organization of such approximations in a spatial access method (SAM).

For selective spatial queries, we need a conservative approximation as a geometric key allowing to identify a superset of the response set and not a subset. Therefore, a progressive approximation can only be stored additionally to a conservative approximation.

A conservative approximation such as the 5-corner can be organized by SAMs using the approximation as geometric key. Some data structures have been designed for special types of approximations. Examples are the cell tree [Gün 89] and the polyhedra-tree [Jag 90a] for convex polygons. From our point of view, these structures are rather complex such that the processing of queries and updates is very CPU-time intensive. Such access methods have to organize complex objects in their directory. This is more difficult than organizing simple rectangles.

Our approach is to manage the geometric key in the R*-tree, a spatial access method originally designed for rectilinear boxes. In several performance analyses and comparisons, the R*-tree has proven its robustness and efficiency. In principle, there are two appealing approaches to managing conservative approximations (e.g. the 5-corner) in the R*-tree:

1.) Store the conservative approximation as a geometric key *instead of* the MBR.
2.) Store the conservative approximation *in addition to* the MBR.

Both approaches require more storage for the approximations; if we store the approximation in addition to the MBR, we need 4 parameters more than in the first approach. The increased storage requirements decrease the capacity of a data page in the R*-tree, and hence worsen its performance. Following the first approach (approximation instead of MBR), there is an additional impact: Except for the convex hull, all of the investigated approximations have a higher x- and y-extension than the MBR-approximation. This yields a higher *area extension* which is computed by the product of the x-extension and the y-extension. An example is depicted in figure 9.

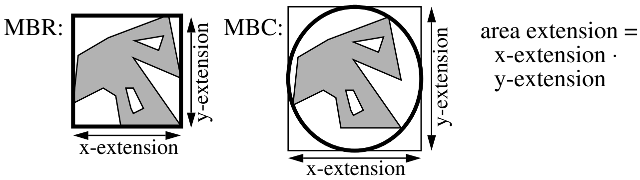


Figure 9: Example for the area extension of an approximation

To get a flavor which area extensions occur in real applications, we computed the area extensions of the real data presented in the former subsections. The experiments show that on the average the area extension of the 5-C is about 21% higher than the area extension of the MBR. The results for the 4-C, the RMBR, and the MBE are 44%, 51%, and 22%, respectively. The area extension determines the size of the corresponding page region. A *page region* is defined by the common MBR of the approximations contained in the associated page. The higher the area of a page region, the worse is the performance of the R*-tree because it is more likely that a query has to access the corresponding page.

Therefore, we observe two contrary effects: approach 1 needs less storage than approach 2, but causes a higher area extension. In order to investigate these impacts, we compared the two approaches with real test data. Each spatial relation consists of about 130,000 objects. These relations were already used in [BKS 93a]. We assume that each description of an object stored in an R*-tree needs 16 Byte for the MBR, 16 Byte for the MER, 20 and 40 Byte for the RMBR and the 5-C, respectively, and 32 Byte for additional information. Page sizes of 2 and 4 KB and an LRU-buffer of 128 KB were used. We tested point queries, window queries, and intersection joins. The window queries were performed with quadratic windows which have an x-extension of 1% and 5% compared to the x-extension of the data space. A join computes about 86,000 pairs of intersecting MBRs. Figure 10 depicts the percentage of page accesses required by approach 2 in comparison to approach 1. If the bar in the diagram is less than 100%, approach 2 is superior to approach 1. Otherwise, approach 1 is superior.

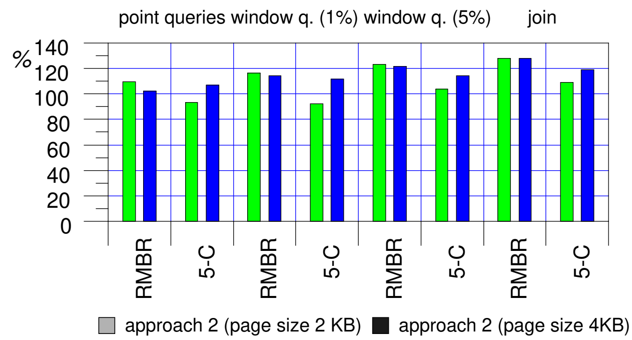


Figure 10: I/O-cost of approach 2 compared to approach 1

Figure 10 shows only slight differences between the two approaches with small advantages for approach 1. For an interpretation of the results, we have also to consider the CPU-cost. The cost for immediately testing a conservative approximation such as the 5-C against a query condition is much higher than for testing the MBR first and then the approximation. One number illustrates this effect: for the intersection join we have to test the approximation using approach 1 about 30 times as often than using approach 2. Additionally, approach 2 is easier to implement and applicable to almost all SAMs. Therefore, it is reasonable to store the conservative approximation in addition to the MBR.

3.5 The performance impact of approximations

Using the same data as in the experiment before and the results from the two preceding subsections, we investigated the change of the total performance of the intersection join storing conservative and progressive approximations in addition to the MBR. Figure 11 depicts the results of our experiments when the MER is used as a progressive approximation and the RMBR or the 5-C is used as a conservative approximation additionally to the MBR. The value ‘loss’ corresponds to the additional page accesses of the MBR-join which are caused by the higher storage requirements. The ‘gain’ is computed under the very cautious assumption that each pair of objects which is identified by the geometric filter as a hit or a false hit, saves the cost of one page access. The ‘total’ value shows the total performance gain using the suggested approximations for the intersection join. The results of figure 11 demonstrate the great performance gains which can be achieved by using adequate approximations. These gains are considerably higher than the additional cost for the MBR-join.

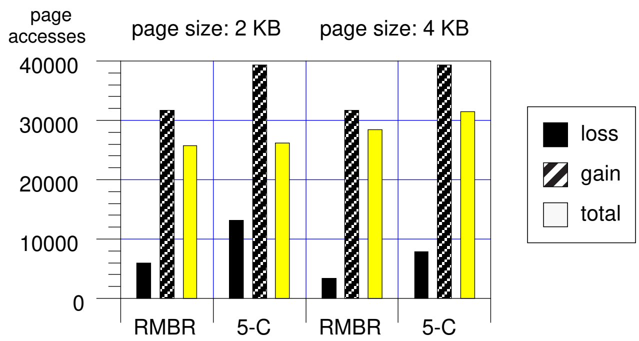


Figure 11: Change of performance using approximations

3.6 Summary

For processing spatial joins, we can summarize that the MBR-filtering can be substantially improved by storing other approximations additionally to the MBR. According to our results, we propose to use the following two techniques: 1.) the 5-corner approximation and 2.) the MER-approximation. For these techniques, the diagram in figure 12 shows the division of the intersecting MBRs into hits and false hits for the test series *BW A*. For the other test series, we have obtained similar results. Altogether, for test series *BW A* we can identify 46% of all intersecting pairs of MBRs as hits or false hits. The remaining pairs are candidates for the exact investigation. In the next section, several techniques are considered for improving the final step of the spatial join processor.

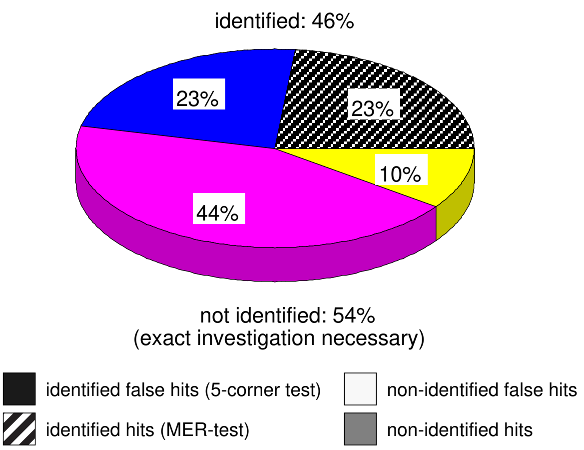


Figure 12: Identified and non-identified hits and false hits of *BW A*

4 Investigation of the Exact Geometry

Geometric filtering is based on the approximations of objects. It identifies hits, false hits and candidates that may fulfill the query condition. The *exact geometry processor* tests whether a candidate actually fulfills the query condition or not. This step is very time consuming and dominates spatial indexing and geometric filtering in many applications [BZ 91]. In this section, we propose methods for reducing the effort for testing the exact geometry. For the intersection join that means to improve the test of intersection.

In the exact geometry processor, we start off with a set of object pairs which may intersect. The straightforward approach to compute whether the exact geometry of a pair of simple polygons (obj_1,obj_2) intersects or not, consists of two steps: in the first step, we search for a pair of edges ($edge_i \in obj_1$, $edge_j \in obj_2$) which intersect. If an intersecting pair of edges exists, then the objects intersect, too. A brute force approach for the first step is to test each edge of obj_1 against each edge of obj_2 which needs $O(n_1 \cdot n_2)$ time where n_1 and n_2 denote the number of edges of obj_1 and obj_2, respectively. If no pair of edges intersects, we have to perform the second step which tests whether obj_1 contains obj_2 or vice versa.

This polygon-in-polygon test can be performed by two point-in-polygon tests. Without preprocessing, such a test needs $O(n_1)$ and $O(n_2)$ time, respectively [PS 85]. The MBR-approximation accelerates the polygon-in-polygon test: Only if the MBR of a polygon obj_1 is contained in the MBR of the other polygon obj_2, obj_1 can be contained by obj_2. Otherwise, we need no point-in-polygon test. For our test data, such MBR-pretests omit between 75% and 93% of the point-in-polygon tests.

Nevertheless, the quadratic cost of the intersection test is not acceptable for real geographic database applications. Therefore, more sophisticated algorithms are absolutely necessary to perform the exact investigation on the object geometry. In the following, we discuss two approaches: a plane-sweep algorithm and an algorithm based on object decomposition.

4.1 Plane-sweep algorithm

Algorithms from the area of computational geometry are proposed to overcome the time bottleneck of operations on the exact object geometry. Different specialized data structures and techniques, such as plane sweep or divide-and-conquer, are used to design efficient algorithms for the different spatial queries and operations. For intersection problems the *plane-sweep technique* is a widely used method [PS 85]. Shamos and Hoey [SH 76] presented a plane-sweep algorithm for detecting an intersection in a set of line segments in $O(n \log n)$ where n is the number of line segments. This algorithm can be easily modified for detecting an intersection between the edges of two polygons in the same worst case time complexity where $n = n_1 + n_2$ (n_1 and n_2 denote the number of edges of the two polygons).

The basic idea is to sort the vertices of both polygons in a preprocessing step according to their x-coordinates. Then a vertical line *(sweep line)* sweeps the data space from left to right. The sweep line stops at every vertex *(event point)*, where the status of the plane-sweep is updated. This *sweep line status* stores the edges which intersect the sweep line and is recorded in a dynamic data structure (e.g. AVL-tree). The edges in the sweep line status are sorted according to their y-coordinate at the sweep line position. For every event point, the corresponding edges are inserted into or deleted from the sweep line status. While the insert operation is performed, the considered edge is tested for intersection with its new neighbors in the sweep line status. For a delete operation the former neighbors of the edge are tested for intersection. If an intersection between two edges from different polygons is detected, the algorithm stops successfully. Otherwise (no intersection exists), the algorithm runs until the last vertex is processed. In consequence, the plane-sweep algorithm is - just as the quadratic algorithm - more expensive for identifying false hits than for identifying hits. Figure 13 depicts an example for the plane-sweep algorithm.

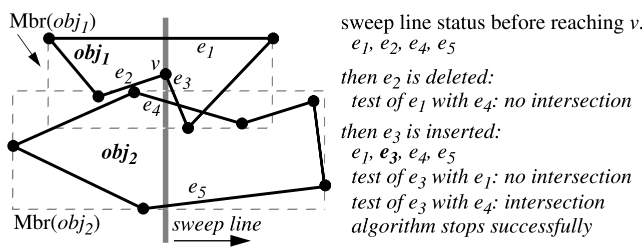


Figure 13: Example for the plane-sweep algorithm

We only have to check edges in the plane-sweep algorithm which intersect the intersection rectangle of the MBRs of the two polygons. In figure 13 for example, the edges e_1 and e_5 do not need to be processed by the plane sweep because they cannot intersect an edge of the other polygon. By a linear scan through each of the two polygons, we can exclude all edges not intersecting this rectangle. This technique is called *restricting the search space*.

4.2 Object representation by decomposition and TR*-trees

Object representation

Another technique for improving the exact geometry processing is based on the following paradigm. In order to support query intensive applications, time and storage is invested in the representation of the spatial objects in order to shift time requirements from query processing to update and restructuring operations. Due to the complexity of the objects and due to the selectivity of spatial queries a *decomposition representation* of the objects is suitable for spatial query processing [KHS 91]. For example, spatial objects can be decomposed into simpler components such as convex polygons [KHS 91], triangles [PS 85], or trapezoids [AA 83] (see figure 14). Such a decomposition approach consists of a single preprocessing step at object insertion time. This preprocessing simplifies spatial query processing because the decomposition substitutes the execution of one complex computational geometry algorithm by multiple executions of fast algorithms applied to simple components. Thus, performing a query on a single complex object is replaced by performing the query on a set of simple objects [KHS 91].

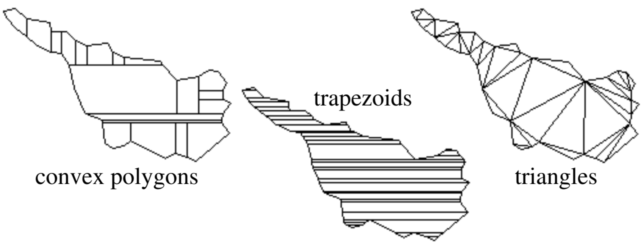


Figure 14: Three decomposition techniques for simple polygons

In the following, we assume that an object is decomposed into trapezoids. The main advantage of trapezoids is that single trapezoids as well as sets of trapezoids can accurately be approximated by MBRs.

TR*-tree

The success of the decomposition approach depends on the ability to quickly narrow down the set of components that are affected by spatial queries and operations. In order to decide which components are relevant for a particular geometric test, we need an efficient data structure that organizes the components of one object with respect to their location.

We use a spatial access method for organizing the components. For this approach, we started our investigations with the R*-tree. However, the R*-tree was designed as a spatial access method for secondary storage. Therefore, we developed the *TR*-tree* [SK 91] which is designed to reduce main memory operations and to store the components of one decomposed object.

The data structure and the algorithms of the R*-tree and the TR*-tree are rather similar: A non-leaf node of a TR*-tree contains entries of the form *(rect, ref)* where *ref* is the address of a child node and *rect* is the minimum bounding rectangle of all trapezoids stored in this child node. A leaf node contains trapezoids *(trap)* as entries. The TR*-tree is persistently stored on secondary storage and is completely transferred into main memory when the complete polygon is required for a geometric operation. In particular, it is not required to build up the TR*-tree in main memory or to convert its pointers.

The height of a TR*-tree grows logarithmically in the number of objects. TR*-trees have to allow overlap in their non-leaf nodes, i.e. rectangles of different entries may have a common intersection. For real geometric objects, the TR*-tree permits nearly logarithmic searching for a point query, but - due to the overlap within its directory - the search is not restricted to one path and thus logarithmic search time cannot be guaranteed. In the worst case, O(n) time is necessary for a point query, where n denotes the number of trapezoids.

The main characteristic of the structure of the TR*-tree is its small maximum number of entries per node which reduces the number of main memory operations. The best performance is obtained by a TR*-tree with a maximum node capacity M of 3 to 5.

The object representation using trapezoids and TR*-trees needs more storage than a representation of an object by point lists. First, the storage of the trapezoids is costlier and second, the directory of the TR*-tree needs additional storage.

Figure 15 depicts the different levels of a TR*-tree that organizes the trapezoids describing the state of Bavaria.

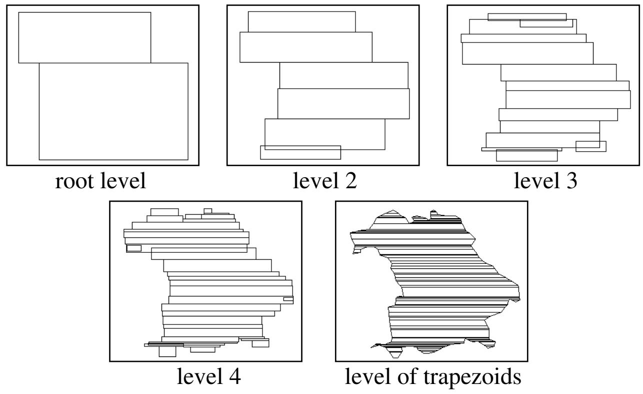


Figure 15: TR*-tree representation of Bavaria

Intersection test

Using two TR*-tree representations, the intersection test between two spatial objects determines whether there exists a pair of intersecting trapezoids or not. The basic idea for such a test is to use the property that the rectangle of an entry in a non-leaf node forms the MBR of the trapezoids in the corresponding subtree. If the rectangles of two entries E_R and E_S do not have a common intersection, there will be no pair *($trap_R$, $trap_S$)* of intersecting trapezoids where $trap_R$ is in the subtree of E_R and $trap_S$ is in the subtree of E_S. Otherwise, there might be a pair of intersecting trapezoids in the corresponding subtrees. This property guides the search through the two TR*-trees (compare [BKS 93a]). When an intersecting pair of trapezoids is found, the algorithm stops successfully. In the worst case, the TR*-intersection test requires O($n_1 \cdot n_2$) time where n_1 and n_2 denote the number of trapezoids of the involved objects.

4.3 Performance comparison

For evaluating the three presented approaches, we need a measure of cost. However, the more sophisticated an algorithm is, the more difficult is it to find a representative and reproducible cost measure. We decided to count the geometric operations in the algorithms. For this approach, two aspects must be observed: First, we have to assign fair weights to the operations. The other aspect is that some parts of the algorithm may not be taken into consideration because not all operations are counted (and weighted).

The first step is to determine which operations should be taken into account. For the quadratic intersection, this is an easy task, because for the first step only *edge intersection tests* and for the point-in-polygon tests only intersection tests between edges of the polygon and auxiliary horizontal lines occur *(edge-line intersection test)*. In the plane-sweep algorithm, two other important operations occur: 1.) In order to determine the position of a new edge in the sweep line status, we must compute the y-coordinate of the new edge at plane-sweep position and compare it to the correspondingly computed y-coordinates of the other edges stored in this status *(position test)*. 2.) When the restriction of the search space is applied, a further important operation is the test whether an edge intersects the intersection rectangle of the MBRs of the tested polygons *(edge-rectangle intersection test)*. Because the sorting of the vertices according to their x-coordinates can be done in a preprocessing step for each polygon, we neglected the cost for this sorting. Contrarily, the TR*-approach is based on two different operations: the *rectangle intersection test* and the *trapezoid intersection test*.

Neither for the plane-sweep algorithm nor for the TR*-approach, we consider the cost for preprocessing. However, the preprocessing cost of the TR*-approach is considerably higher than the preprocessing cost of the plane-sweep algorithm. In consequence, the TR*-approach does not pay off for performing a low number of intersection tests.

In order to determine weights for these operations, we measured the time required for performing them on an HP720-workstation. Table 6 depicts the obtained weights. For the plane-sweep algorithm (without sorting of the vertices according to their x-coordinates) the four counted operations need about 90% of the total computing time; for the other approaches, the time represented by the counted tests is even higher.

operation	weight (in 10^{-6} sec)
edge intersection test	15
edge-line intersection test	18
position test	36
edge-rectangle intersection test	28
rectangle intersection test	28
trapezoid intersection test	38

Table 6: Weights for the different geometric operations

For our comparison, we selected the test series from section 3 as test data. After performing the geometric filter with the 5-corner-test and the MEC-test, we obtain a set of candidate pairs which are transferred into main memory. In table 7, we present the cost of the three investigated algorithms for the *Europe A* and the *BW A* test series. In this table, we present the results of the plane-sweep with restricting the search space because this version saves about 40% of the cost compared to the version without restricting the search space. The TR*-tree algorithm is performed with a maximum node capacity of 3 entries. For the test series *Europe B* and *BW B,* we obtain similar results.

test series	algorithm	intersections (hits)		non-inters. (false hits)		total cost
		#	cost per pair	#	cost per pair	
Europe A	quadratic	867	119.6	197	154.3	164,193
	plane-sweep		9.9		10.9	10,732
	TR*-tree		0.7		1.0	795
BW A	quadratic	1,026	2814.7	223	7487.8	4,557,686
	plane-sweep		49.2		51.6	62,024
	TR*-tree		0.9		1.3	1,263

Table 7: Cost of the exact intersection algorithms (in 10^{-3} sec)

One result is obvious: the quadratic algorithm is - as expected - extremely inferior to the other algorithms and needs no further discussion. The first observation for the plane-sweep algorithm is that the cost for determining a non-intersection is nearly the same as the cost for determining an intersection. This effect results from the fact that we restrict the search space. Without restricting the search space, the identification of a false hit is costlier by a factor of about 2.3. In our test relations, the average number of edges per polygon is 84 for *Europe A* and 527 for *BW A*. The higher number of edges in *BW A* causes considerably higher cost for processing one object pair in *BW A* than one in *Europe A*. The dependence between the number of edges and the cost of the plane-sweep algorithm is represented in the upper diagram of figure 16. This diagram depicts the cost to determine a false hit between two polygons in the *BW A* test series depending on the number of edges of both polygons. The diagram demonstrates the strong dependency between the cost and the number of edges.

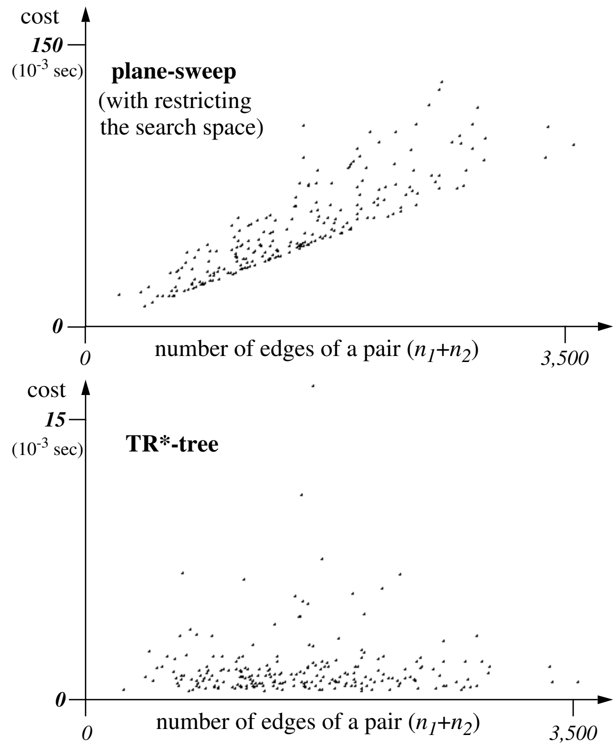


Figure 16: Cost of intersecting a pair of polygons *(BW A)*

The TR*-tree algorithm behaves quite different. The most important observation is the low dependency between the average number of edges and the cost of the algorithm. According to table 7 the cost per object pair differs only by a factor of 1.35 between *BW A* and *Europe A*. This ratio is smaller than the ratio of the average heights of the TR*-trees which is $^{7.6}/_{5.0}$ = 1.52. Considering the lower diagram of figure 16, we see that for an individual pair of polygons other properties (presumably their overlap) influence the performance of the TR*-tree algorithm. In test series *BW A* with 527 edges per polygon, the algorithm needs on the average 33.2 rectangle-intersection-tests and 2.2 trapezoid-intersection-tests for identifying a hit. These are pleasantly small numbers which explain the extremely good performance of the TR*-tree approach. In our experiments, we achieve high performance improvements by at least one order of magnitude compared to the plane-sweep algorithm. Therefore, the TR*-tree algorithm is from our point of view the best choice, even for objects with a lower complexity.

Another interesting result is that the TR*-tree algorithm has the best performance with a maximum node capacity *(M)* of 3. If higher node capacities are used, the number of rectangle intersection tests as well as the number of trapezoid intersection tests is increasing.

The cost of using TR*-trees with different node capacities is shown for test series *BW A* in figure 17. The fact that the number of trapezoid intersection tests is lowest for the smallest maximum node capacity can be expected. However, that also the number of rectangle-intersection-tests is minimal, is a little bit surprising, because with lower node capacities the heights of the TR*-trees are increasing and the possibilities for an efficient space partitioning are decreasing. But these effects are compensated by the lower search cost in nodes with a very small capacity.

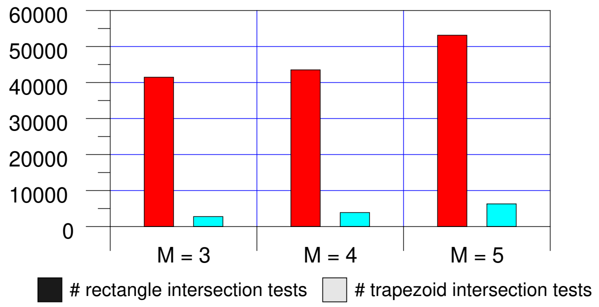


Figure 17: Performance of TR*-trees with different max. capacities

In spite of the problems that occur when different types of algorithms are compared, we summarize that the naive quadratic approach is out of question and that the TR*-tree approach clearly outperforms the plane-sweep algorithm. The TR*-tree achieves high factors of performance improvement and additionally benefits from its clear algorithmic concepts which simplify the implementation of this technique.

5 Comparison of the Total Performance Gains

In this section, we summarize the impacts of the presented techniques with respect to the total execution time of an intersection join. We tried to be as fair as possible, however, the reader should be aware that the following results only give a tendency of the cost.

We join two spatial relations of real world objects each consisting of about 130,000 objects. As a result, 86,000 pairs of MBRs intersect. We assume that each description of an object stored in an R*-tree needs 16 Byte for the MBR, 16 Byte for the MER, 40 Byte for the 5-C, and 32 Byte for additional information. Page size is 4 KB and an LRU-buffer holds 32 pages of the R*-tree. The CPU-time spent in the second step for testing the intersection of two approximations is neglected. Furthermore, we assume that each pair of objects which is not identified by the geometric filter as a hit or a false hit, causes the cost of one page access (= 10 msec) and that the TR*-tree representation increases the access cost for an investigated object by a factor of 1.5. This is because the TR*-tree representation has a higher storage cost than a representation by simple point lists. The cost for the exact investigation of one object is 25 msec for the plane-sweep algorithm and 1 msec for the TR*-tree algorithm. These numbers are averages obtained from our experiments in the previous section. Figure 18 depicts the results of our investigation.

Version 1 is the starting point of our investigation. It uses no additional approximations and it uses the plane-sweep algorithm for the exact intersection test. The figure demonstrates that the cost for transferring the object into main memory (object access) and for the exact intersection test dominates the total execution time. In version 2, the 5-C and the MER-approximation are introduced. These additional approximations considerably decrease the cost for the object access and the cost for the exact test, but increase the cost for the MBR-join. However, the total execution time is roughly reduced by 40%. In version 3, the plane-sweep algorithm is replaced by the TR*-tree algorithm. Now, the execution time for the exact

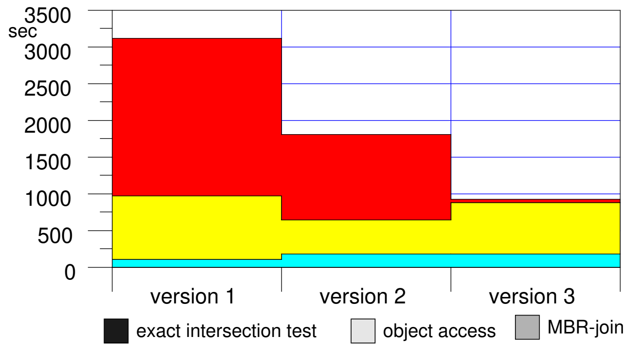


Figure 18: Total join performance using different techniques

investigation is practically negligeable but the cost for the object access increases. Nevertheless, the total execution time is improved by a factor of almost 2 compared to version 2, and by a factor of more than 3 compared to version 1.

6 Conclusions

In this paper, we investigated the problem of efficient spatial join processing on two spatial relations. The spatial join is one of the most important and time consuming operations in a spatial database system. Therefore, an efficient execution is crucial to its overall performance. Our investigation uses the intersection predicate which tests whether an object of the one relation intersects an object of the other relation. The presented results are obtained from experimental performance comparisons using real cartographic data. Due to space limitations, we restricted our presentation to a few of our experiments. Without exceptions, other experiments showed similar results and confirmed to use our approach for spatial join processing.

Our basic idea for efficient spatial join processing is to use several filter steps. In a filter step, so-called candidates are computed, i.e. pairs of spatial objects which may fulfill the join predicate. Moreover, a large number of the candidates can already be identified as answers (hits) or as false hits.

In the first step, the spatial join is performed on the minimum bounding rectangles (MBRs) of the objects. This operation is called MBR-join. The MBRs serve as a geometric key to support fast access to the objects using a spatial access method (SAM). In [BKS 93a], we have already proposed an efficient algorithm for computing MBR-joins.

The second step of join processing is the geometric filter. The geometric filter detects a high fraction of false hits using conservative approximations in addition to the MBR. According to a detailed experimental investigation, we recommend to use the minimum bounding 5-corner of objects as an additional approximation. The 5-corner combines a high accuracy with a small storage overhead. In our experiments, about two thirds of the false hits in the candidate set computed by the MBR-join are identified using the 5-corner. In order to detect hits in the candidate set (still without accessing the exact geometry), we propose to use simple progressive approximations, e.g. enclosed circles and enclosed rectangles. A common intersection of two progressive approximations implies a common intersection of the corresponding objects. In our experiments, about one third of the hits are detected using the enclosed rectangle as a progressive approximation.

In the third step, the join predicate has to be checked for all remaining candidates using the exact geometry. A straightforward and very inefficient solution for the intersection test of polygonal objects is an algorithm which basically tests each pair of edges for intersection. A more efficient approach is to use the plane-sweep technique well-known from the area of computational geometry.

We suggest a new approach for join processing which decomposes polygonal objects into sets of trapezoids. For each object, a TR*-tree - a main-memory resident variant of the R*-tree - organizes the corresponding trapezoids. In several experiments, we demonstrated that the TR*-tree approach improves the computation of the join predicate by at least one order of magnitude in comparison to the plane-sweep approach. This effect demonstrates the importance of an adequate main memory representation of objects for database systems. The more complex the object, the more significant is the quality of the object representation.

Using additional conservative and progressive approximations as well as the TR*-tree approach, the total execution time of the spatial join is improved by a factor of more than 3 for our test data. Our results showed that the MBR-join does not much affect the total execution time of the spatial join. In particular, this can be observed when the third step is performed using the plane-sweep approach. One reason is that our approach for processing MBR-joins is already very efficient. Due to our elaborated representation of objects using TR*-trees in main memory, the time spent for testing the join predicate also does not much influence the execution time anymore.

Since we reduced the consumption of CPU-time essentially, the major cost factor in the final version of our join processor is the time spent for fetching objects from disk into main memory. This result demonstrates the necessity for supporting an efficient access to complex spatial objects stored on secondary storage. Especially for spatial and non-spatial joins performed on complex join attributes, this will be an important area for future work [BK 94].

So far, we assume a conventional computer architecture. However, since the fast execution of spatial join processing is extremely important, another task is to consider CPU- and I/O-parallelism in future work.

References

[AA 83] Asano Ta., Asano Te.: *'Minimum Partition of Polygonal Regions into Trapezoids'*, Proc. 24th IEEE Annual Symp. on Foundations of Computer Science, 1983, pp. 233-241.

[BG 90] Blankenagel G., Güting H.: *'Internal and External Algorithms for the Points-in-Regions Problem - the INSIDE Join of Geo-Relational Algebra'*, Algorithmica, 1990, pp. 251-276.

[BHKS 93] Brinkhoff T., Horn H., Kriegel H.-P., Schneider R.: *'A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems'*, Proc. 3rd Int. Symp. on Large Spatial Databases, Singapore, 1993, Lecture Notes in Computer Science, Vol. 692, Springer, pp. 357-376.

[BK 94] Brinkhoff T., Kriegel H.-P.: *'The Impact of Global Clustering on Spatial Database Systems'*, Technical report, University of Munich, 1994.

[BKS 93a] Brinkhoff T., Kriegel H.-P., Seeger B.: *'Efficient Processing of Spatial Joins Using R-trees'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Washington, DC, 1993, pp. 237-246.

[BKS 93b] Brinkhoff T., Kriegel H.-P., Schneider R.: *'Comparison of Approximations of Complex Objects used for Approximation-based Query Processing in Spatial Database Systems'*, Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 40-49.

[BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

[BZ 91] Benson D., Zick G.: *'Symbolic and Spatial Database for Structural Biology'*, Proc. Conf. on Object-Oriented Programming Systems, Languages, and Applications, Phoenix, AZ, 1991, pp. 329-339.

[DB 83] Dori D., Ben-Bassat M.: *'Circumscribing a Convex Polygon by a Polygon of Fewer Sides with Minimal Area Addition'*, Computer Vision, Graphics, and Image Processing, Vol. 24, 1983, pp. 131-159.

[Fal 88] Faloutsos C.: *'Gray Codes for Partial Match and Range Queries'*, IEEE Trans. on Software Engineering, Vol. 14, No. 10, 1988, pp. 1381-1393.

[Gün 89] Günther O.: *'The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases'*, Proc. 5th Int. Conf. on Data Engineering, Los Angeles, CA, 1989, pp. 508-605.

[Gün 93] Günther, O.: *'Efficient Computations of Spatial Joins'*, Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 50-59.

[Gut 84] Guttman A.: *'R-trees: A Dynamic Index Structure for Spatial Searching'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, MA, 1984, pp. 47-57.

[Jag 90a] Jagadish H. V.: *'Spatial Search with Polyhedra'*, Proc. 6th Int. Conf. on Data Engineering, Los Angeles, CA, 1990, pp. 311-319.

[Jag 90b] Jagadish H. V.: *'Linear Clustering of Objects with Multiple Attributes'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 332-342.

[KBS 93] Kriegel H.-P., Brinkhoff T., Schneider R.: *'Efficient Spatial Query Processing in Geographic Database Systems'*, IEEE Data Engineering Bulletin, Vol. 16, No. 3, 1993, pp. 10-15.

[KHS 91] Kriegel H.-P., Horn H., Schiwietz M.: *'The Performance of Object Decomposition Techniques for Spatial Query Processing'*, Proc. 2nd Symp. on Large Spatial Databases, Zurich, Switzerland, 1991, Lecture Notes in Computer Science, Vol. 525, Springer, pp. 257-276.

[ME 92] Mishra P., Eich M. H.: *'Join Processing in Relational Databases'*, ACM Computing Surveys, Vol. 24, No. 1, 1992, pp. 63-113.

[Ore 86] Orenstein J. A.: *'Spatial Query Processing in an Object-Oriented Database System'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Washington, DC, 1986, pp. 326-333.

[PS 85] Preparata F. P., Shamos M. I.: *'Computational Geometry'*, Springer, 1985.

[Sam 90] Samet H.: *'The Design and Analysis of Spatial Data Structures'*, Addison-Wesley, 1990.

[SH 76] Shamos M. I., Hoey D. J.: *'Geometric Intersection Problems'*, Proc. 17th Annual Conf. on Foundations of Computer Science, pp. 208-215, 1976.

[SK 91] Schneider R., Kriegel H.-P.: *'The TR*-tree: A New Representation of Polygonal Objects Supporting Spatial Queries and Operations'*, Proc. 7th Workshop on Computational Geometry, Bern, Switzerland, 1991, Lecture Notes in Computer Science, Vol. 553, Springer, pp. 249-264.

[SRF 87] Sellis T., Roussopoulos N., Faloutsos C.: *'The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects'*, Proc. 13th Int. Conf. on Very Large Databases, Brighton, England, 1987, pp 507-518.

[Wel 91] Welzl E.: *'Smallest Enclosing Disks (Balls and Ellipsoids)'*, Technical report B91-09, Freie University of Berlin, 1991.