Query optimization for GIS using filters

Hein M. Veenhof
veenhof@cs.utwente.nl

Maurice A.W. Houtsma [*]
houtsma@cs.utwente.nl

Peter M.G. Apers
apers@cs.utwente.nl

University of Twente
Department of Computer Science
P.O. Box 217, 7500 AE Enschede
the Netherlands

Abstract

When viewing present-day technical applications that rely on the use of database systems, one notices that new techniques must be integrated in database management systems to be able to support these applications efficiently. This paper views one of these techniques in the context of supporting a Geographic Information System. For efficient retrieval of geometric data, we show that queries can be optimized by filtering data not with just one but with several simple filters. A prototype of a query optimizer/evaluator using this new technique is described together with preliminary test results.

1 Introduction

In the past, much research has been done on query optimization techniques for (relational) databases [1]. Optimization efforts mainly concentrated on queries stemming from administrative applications. Recently, databases are increasingly used not only for administrative, but also for technical applications. Technical applications, e.g. CAD/CAM systems, GIS, and multimedia systems, place a heavy burden upon a database management system. This is caused by aspects such as huge amounts of data, complex data structures, and the relative importance of other operations. Because of the different characteristics of technical applications, query optimization techniques need to be reconsidered. Also, new techniques should be developed, especially for technical applications.

To reduce the heavy burden that operations in technical applications place on the database management system, *filters* are used [3]. A filter acts as a preprocessor for an operation. The main idea of filters is to reduce the size of the operands. Thus, a filter is used before an operation to reduce the size of its operands (just like semi-joins are used to reduce join-operands in relational queries [2]). With smaller operands, the cost of an operation will be smaller; however, the cost of using a filter has to be taken into account. In addition to using one filter, several filters may be combined into a filter sequence. Each filter in such a sequence will reduce the operand in size or simplify it.

We consider a Geographic Information System (GIS) as a typical technical application. The important operations to be supported, differ from those in traditional database systems and some are very expensive (for instance, overlay and intersection). A GIS typically maintains thematic data, (e.g., street names, soil type, area size), and geometric data, (e.g., geometry of buildings, of land parcels, of mountains). Thematic data can easily be supported by a database. The advantages of databases, amongst others persistency, efficient retrieval, and recovery, should also be exploited for geometric data.

The optimization techniques developed to access geometric data concentrate on spatial indices [5, 10]. Here, we do not concentrate as much on indices, as well as on filters. An important reason for this, is that we expect queries in a GIS to be composed of many spatial joins (n-way joins). We cannot expect a spatial index to be present on every intermediate result, so it is worthwhile studying techniques that go beyond a spatial index. We propose a generic model for query optimization using filter techniques.

2 Using filters in optimization

The spatial join, e.g., required for overlay operations, is one of the most expensive database operations in a GIS. Various spatial indices have been developed to decrease response times of spatial joins. We will not discuss these indices here. Instead, we focus on filters. An example of a spatial query composed of many spatial joins is:

[*] The research of Maurice Houtsma has been made possible by a fellowship of the Royal Netherlands Academy of Arts and Sciences

```
Retrieve all rural areas below sea level
         having soil type equal to sand
         within 3 miles of polluted lakes
```

With layers land use, soil, pollution, and elevation, several spatial joins have to be calculated to construct the answer to this query. A spatial join of, e.g., layers land use and elevation, gives the intermediate result to the subquery: `rural areas below sea level`. It are those intermediate results, to be used as operands of successive spatial joins, that can effectively be reduced in size by the use of filters. Building spatial indices for these intermediate results is considered too expensive.

Without using indices, a spatial join of two relations can be solved by calculating the spatial comparison operation for each pair of spatial objects in a nested-loop strategy. For two spatial relations R and S with cardinality n and m respectively, the spatial comparison operation Θ is evaluated $n * m$ times. To speed up the evaluation process, a simple test can be performed before evaluating the actual comparison operation. This test indicates if, for a given pair of spatial objects, the comparison operation can be avoided altogether.

An example of such a simple test, in case of e.g. an intersection operation, is whether the bounding boxes of the spatial objects have any overlap at all. If they have no overlap, then there is no need to calculate the overlap of the spatial objects themselves. The bounding box of a spatial object is an example of a simplification of the spatial object, often called approximation [5, 9]

So, when using a filter, instead of evaluating Θ, a simple test operation f_Θ is performed for each pair of object approximations. Operations on approximations are much cheaper than on the spatial objects themselves. Following a nested-loop strategy, this test is performed $n * m$ times, resulting in a set of k candidate pairs (with $k \leq (n * m)$). After the test has been performed, Θ now has to be evaluated only k times.

We can extend this approach to using a sequence of filters. The query optimizer, based on cost estimations, can decide to use multiple filters in a sequence; each one reducing the set of candidate pairs further. This is shown in Fig. 1. A sketch of the architecture of the query optimizer is given in Fig. 2.

The benefit of using a sequence of filters can be estimated as follows. Assume a sequence of filters $F_1, \ldots, F_n$, with a resulting set of k_n candidate pairs. The cost of the remaining spatial comparison operation is $k_n C_\Theta$, plus the cost of reading the k_n-pairs from disk and writing the output back to disk. The cost of applying the filters is given by $C_{k_1} + \ldots + C_{k_n}$, where each C_{k_i} is a function of the cost of the test operation and the number of candidate pairs resulting from F_{i-1}.

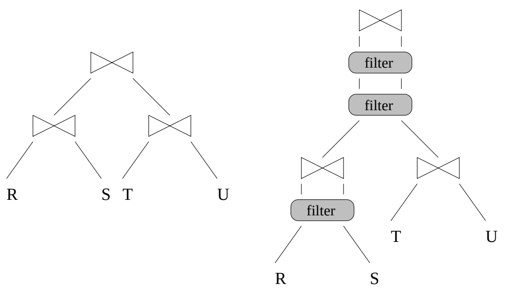

Figure 1: Query tree and its optimized version.

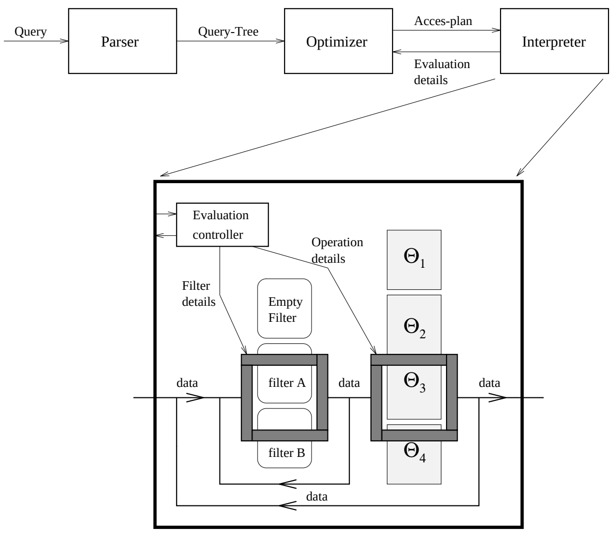

Figure 2: Architecture of (dynamic) query optimizer using filters.

Not only may the use of filters result in a smaller number of CPU-intensive spatial comparison operations, it may also result in considerable savings on the number of disk accesses. This, because approximations will occupy considerable less space than the spatial objects. Of course, the query optimizer also has to take into account the availability of spatial indices on approximations.

We assume that the query optimizer generates a query tree based on its particular cost model. For each filter included in the query tree, it indicates the expected number of objects that will result after applying the filter. If, during query evaluation the actual number of candidate pairs differs significantly from the estimation, new filters may be included in the query tree. Because spatial operations are very expensive and we expect estimations concerning spatial queries to be less accurate than estimations for standard queries, dynamic query optimization may actually pay off; whereas in relational systems it is usually considered too complex and expensive.

3 Implementation overview

We are currently implementing a prototype of the query optimizer/evaluator in C++ using the library LEDA [4]. It allows us to check the effectiveness of filter techniques, and the usefulness of using several filters instead of just one.

The evaluator part of the prototype is in its present state able to calculate the overlay of two maps using several or none of the implemented filters. Among these filters are: the *minimum bounding box* (MBB) which is the well known axes-parallel rectangle fitted along the boundary of the object approximated. The *double minimum bounding box* (DMBB), a derivative of the MBB, where both boxes have the same center but the area of the DMBB is about 50% larger. It is introduced to see whether rough approximations, which sometimes can be constructed almost for free, can be useful as part of a filter sequence. The *45 degrees rotated bounding box* (RBB) which is in fact an axes-parallel MBB of the rotated object. And finally, the *minimum bounding circle* (MBC) [6]. Other filters not realised yet are, e.g., *convex hull* [7], and *n-corner*. An n-corner is defined as the optimal n-sided polygon circumscribing a convex polygon [8].

As stated before, the test operation f_Θ must be simple compared to its associated spatial comparison operation Θ. In the case of the three box filters MBB, DMBB, and RBB, the test operation is a box intersection test. At most four coordinate comparisons are used for this test. For the MBC, the test operation is the circle intersection test. Two circles intersect if the distance between their centers is smaller than the sum of their radii. The circle intersection test is more expensive than the box intersection test, since it involves calculating a distance instead of simple coordinate comparisons. Intersection tests for convex hulls and n-corners are even more expensive.

We implemented the overlay with a simple nested-loop join algorithm (see Figure 3) adapted to the use of successive filters. One of the maps is designated as the inner map, and the other as the outer map. For each geometric object of the outer map, all objects of the inner map are read and compared with the object from the outer map. These comparisons are, of course based on the filter techniques described in the previous section. Whenever the join condition is satisfied the intersection of the two objects is calculated and an intersection is placed in the result map.

To get a feeling of the behavior of each filter, we decided to look primarily at the reduction in the number of candidates they can achieve. Furthermore, we looked at CPU costs of test operations, intersection operations, and creation of approximations. We ran tests on objects already located in main memory; costs for accessing the disk have not been taken into account

```
for each object $a$ in map $A$ do
  { for each object $b$ in map $B$ do
     { repeat
          get approximation $a'$ from $a$, and $b'$ from $b$;
          take filter $F$ from filter sequence $F_{seq}$;
          detect overlap by calculating $f_\Theta(a', b')$;
       until ($F_{seq}$ has been exhausted or
              no overlap is detected);
       if overlap detected
       then { determine intersection of objects $a$ and $b$;
              place the result (if any) in map $C$ } } }
```

Figure 3: Nested-loop join algorithm for calculating the overlay of two maps adapted to the use of multiple filters.

in the current version of the prototype.

The implementation is such that approximations needed by a filter are created and stored in memory the first time a filter is used. When an approximation is needed again, it is already available in memory. It is also possible to precalculate all of the approximations and store them in main memory before the actual test-run. This enables us to determine the cost of constructing approximations during the query evaluation.

4 Results

Some preliminary results of the tests are shown in Table 1 and Table 2 for the overlay operation. They show that each filter has its own precision; note, e.g., that the minimum bounding box (MBB) filter reduces the number of candidate pairs more than the double minimum bounding box (DMBB) filter, which was to be expected. Results also show that a sequence of filters may indeed reduce the number of candidate pairs more than a single filter. For instance, MBB followed by 45 degrees rotated bounding box (RBB) reduces the number of candidate pairs from 263 to 238 (see Table 1). This also resulted in a 10% quicker response of the query compared to only using MBB's. This gain was only noticed when all approximations were precalculated. In particular, when minimum bounding circles had to be made during the test-run these constructions dominated the run-time. Using more filters is no guarantee for an extra reduction in the number of candidate pairs. For the first overlay there is no reduction at all when an extra MBC filter is used after the filters MBB, and RBB (567 versus 567 candidate pairs in Table 2). A further reduction for this example is still possible since the overlay operation is called 123 times too many.

The test presented in both tables were done on real world and randomly generated data. For each generated map we produced a number of convex polygons without holes. We then performed an overlay of two maps, to give us all objects formed by the intersection of an object on map A with an

filter(s) used	no of cand. pairs
none	25750
DMBB	542
MBB	263
MBB, RBB	238
MBB, MBC	246
MBB, RBB, MBC	236

Table 1: Result of using (multiple) filters on number of remaining candidate pairs. Overlay of two maps with artificial data.

filter(s) used	no of cand. pairs
none	60000
DMBB	1442
MBB	772
MBB, RBB	567
MBB, MBC	760
MBB, RBB, MBC	567

Table 2: Result of using (multiple) filters on number of remaining candidate pairs. Overlay of two maps with real world data.

object on map B. For the results of Table 1 we used a map of Europe containing 515 polygons, each having between 4 and 1932 vertices. The other map in this test contained 50 convex polygons of which none had more than 30 vertices. See also Figure 4 for a part of these maps and the resulting overlay. For the second test of Table 2 we used two maps containing 150 and 400 generated objects, respectively. The first map only contained polygons with four vertices, while the second map was filled with triangles.

5 Conclusions

The technique of using multiple filters in a row to reduce the number of calls of an expensive operation can be beneficial when certain conditions are satisfied. The bounding box filter is already very good and responsible for a huge gain in response time; it should be used as the first filter in a sequence. Test operations must be very simple in comparison to the operation replaced; nothing is gained when expensive filters are used to avoid calculating a relatively simple operation. A cheap test is preferred to a costly test for obtaining the same answer, and a filter with a test operation being more expensive than another must precede the latter one in the filter sequence. Successive filters should use approximations fitting objects increasingly narrow, avoiding that the filters at the end of the sequence have no filter effect at all. Approximation data needed by the test operations must either be available when needed or it must be cheap to calculate them on the spot. Calculating bounding circles, or even convex hulls and n-corners, during the query

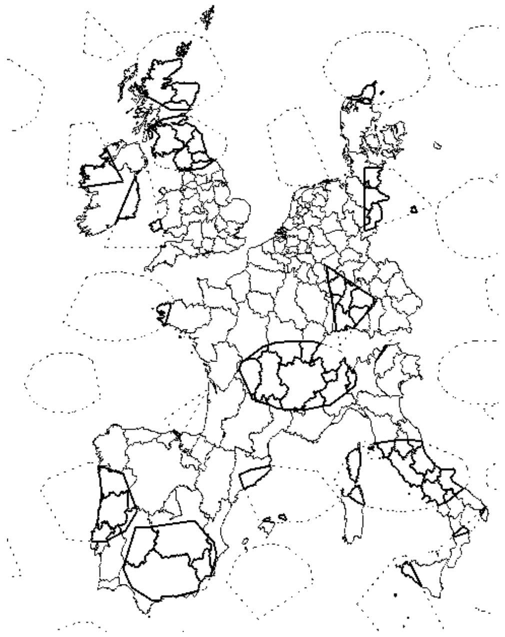

Figure 4: Real world test data combined with generated polygons.

evaluation process is too costly. Especially, when an approximation of an intermediate result is needed, deriving such data from the approximation of the objects it was built from, is, although rough, much cheaper in the end. For example, a rough approximation of the intersection of two simple polygons is the intersection of their bounding boxes.

6 Future work

In the future we will extend the prototype with better fitting approximations, e.g., convex hull, and, n-corner. We will also try to come up with several simple filters. Building approximations from existing approximations is also a topic of further research. A comparison must be made between building a new spatial access tree for an intermediate result or using filters. Further, we must keep in mind that disk accesses for objects could play a major role in technical applications. Therefore, we must also take into account costs for disk-I/O. Extensive testing should bring more insights in the specific aspects of this multi-filter approach. We are now running more tests on real world spatial data. A report of these tests will appear in a full version of this paper.

References

[1] Jarke, M. and Koch, J., “Query optimization in database systems”, *Computing Surveys*, 16(2):111–152, June 1984.

[2] Ceri, S. and Pelagatti, G., *Distributed databases*, McGrawHill, 1984.

[3] Orenstein, J.A., “Redundancy in spatial databases”, *Proc. SIGMOD 89.*

[4] Näher, S., “LEDA User Manual, version 3.0,” Max-Planck-Institut für Informatik, Saarbrücken, Germany.

[5] Brinkhoff, T. and Kriegel, H-.P. and Seeger, B., “Efficient Processing of Spatial Joins Using R-trees”, *Proc. SIGMOD 93.*

[6] Skyum, S., “A simple algorithm for computing the smallest enclosing circle”, *Information Processing Letters*, 37(1991):121–125, February 1991.

[7] Preparata, F.P. and Shamos, M.I. , *Computational Geometry; an introduction*, Springer–Verlag, 1985.

[8] Dori, D. and Ben–Bassat, M., “Circumscribing a convex polygon by a polygon of fewer sides with minimal area addition”, *Computer Vision, Graphics and Image Processing*, 24(1983):131–159, 1983.

[9] Brinkhoff, T. and Kriegel, H-.P. and 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.

[10] Samet, H., “Spatial Data Structures,” To appear in *Database Challenges in the 1990’s*, W. Kim, ed., Addison Wesley/ACM Press, Reading, MA, 1994.