Early Separation of Filter and Refinement Steps in Spatial Query Optimization*

Ho-Hyun Park† Chan-Gun Lee† Yong-Ju Lee‡ Chin-Wan Chung†

†Department of Computer Science, KAIST, Korea
{hhpark, cglee, chungcw}@islab.kaist.ac.kr

‡Department of Computer Engineering, Sangju National University, Korea
yjlee@computer.sangju.ac.kr

Abstract

The spatial query has been processed in two steps, the filter step and the refinement step, due to a large volume and high complexity of the spatial data. However, this approach has been considered only in the query execution phase after completing the query optimization phase. This paper presents query optimization strategies which take the characteristics of spatial databases into account. The first strategy is the separation of filter and refinement steps not in the query execution phase but in the query optimization phase. As the second strategy, several refinement operations can be combined in processing a complex query, and as the third strategy several filter operations can also be combined. We call the optimization technique utilizing these strategies the Early Separated Filter And Refinement (ESFAR). This paper also presents a rule-based optimization technique for ESFAR.

1. Introduction

For the past several years, the research on spatial database systems has actively progressed because the applications using the spatial information such as geographic information systems (GIS), computer aided design (CAD) and multimedia systems, have increased. However, most of the research has dealt with only a part of spatial database systems such as data models, spatial indexes, spatial join algorithms, or cost models. There has been a little research on the spatial query optimization which can integrate them. Most of the spatial query optimization techniques published until now have not properly reflected the characteristics of the spatial databases.

Spatial databases have the following characteristics compared with traditional relational databases or object-oriented databases. First, spatial databases store non-spatial data as well as spatial data, and a query in the spatial databases is a mixed query which contains both spatial subqueries and non-spatial subqueries. Second, the processing cost of the spatial query is very expensive because spatial data is more complex and larger than non-spatial data. Therefore, the spatial query has been processed mostly in two steps, the *filter step* and the *refinement step* [11]. Third, most spatial databases have spatial indexes for spatial data types [5] and the effect of the spatial indexes is bigger than that of non-spatial indexes. The two-step processing of the spatial query reduces not only the CPU time but also the I/O time because the two-step processing makes it possible to obtain the object identifiers for the candidate objects by accessing only the spatial index.

Several spatial database systems have addressed the optimization problem for the spatial and non-spatial mixed query in the literature [10, 2, 1]. An excellent survey is presented in [14]. However, none of the above optimizers provides the filter and refinement steps for spatial operators as individual operators. Therefore, the above optimizers cannot generate the plans which will be proposed in this paper.

This paper presents query optimization strategies which take the characteristics of spatial databases into account. The first strategy to be presented is an early separation of the filter and refinement steps, which means the separation is actually done in the query optimization phase instead of the query execution phase. When an input query consists of a spatial subquery and a non-spatial subquery and there is a spatial index for the spatial subquery, the processing or-

*This research was supported by the National Geographic Information Systems Technology Development Project and the Software Technology Enhancement Program 2000 of the Ministry of Science and Technology of Korea.

der of "filter step – non-spatial operation – refinement step" can be more efficient than that of "non-spatial operation – spatial operation" or "spatial operation – non-spatial operation." As the second strategy, several refinement operations can be combined in processing a complex query, and as the third strategy several filter operations can also be combined. We use the select-merge rule[1] of relational algebra optimization rules for combining refinement steps [15], and the *Oid-intersection* technique [9] and the *Oid-join* technique [4, 16, 6] for combining filter steps. We call the optimization technique utilizing these strategies the *Early Separated Filter And Refinement (ESFAR)*.

This paper also presents a rule-based optimization technique for ESFAR. The input query of a rule-based optimizer is in an algebraic form. In this paper, we use the *Spatial Object Algebra (SOA)* [12] to represent the input query of our optimizer. In addition, we need a new object algebra for ESFAR which separates the operators in SOA into filter step operators and refinement step operators. We define the *Intermediate Spatial Object Algebra (ISOA)* as the new object algebra. Using ISOA, we derive some optimization rules for ESFAR. We implemented the ESFAR optimizer. The Volcano optimizer generator (VOG) was used as a development tool of our optimizer. Through experiments, we compare the ESFAR optimization technique with a traditional optimization technique which does not separate filter and refinement steps. The experimental results show that the ESFAR optimization technique generates more efficient query execution plans than the traditional one.

The remainder of this paper is organized as follows. In Section 2, we summarize some previous studies which are regarded as the background for our work. In Section 3, we explain the optimization strategies which separate the filter and refinement steps early. The ISOA and optimization rules for ESFAR are explained in Section 4. The implementation and experimental results using VOG are shown in Section 5. In Section 6, we conclude this paper.

2. Backgrounds

2.1. Join Algorithms Using Indexes

There are many join algorithms which use the index of both join inputs. There are also many types of indexes. In this paper, however, we consider the join algorithms that use only the R*-tree and the B+-tree which are fairly efficient and the most popular.

In the spatial join area, when R*-trees exist for both join inputs, [5] proposed a join algorithm which synchronously traverses both R*-trees by the depth-first search. This algorithm uses a local optimization policy to fetch the MBR-pairs of child nodes. Later, the algorithm was improved to accomplish the global optimization by the breadth-first search [8]. In this paper, we call both of the join algorithms the *Rtree-join*.

In a non-spatial join, if B+-trees exist for both join inputs, a *Btree-join* similar to the Rtree-join is possible. The Btree-join synchronously traverses only the leaf nodes of both B+-trees by the merge-join technique. The Btree-join technique was already used in [4], and was especially efficient when both indexes were clustered.

2.2. Oid-Intersection and Oid-Join

When multiple indexes exist in a single class and a conjunctive query is issued by the user, *index intersection* technique can be applied [9]. The main idea of the index intersection technique focuses on the *Oid-intersection* between the oid lists resulting from each index probing.

When a query consists of the select and join operations and indexes exist for all join attributes as well as selection attributes, the query can be evaluated by using indexes and oids [4, 16, 6]. This can be performed by a sequence of the natural join between the oid-tuple lists which were obtained from the Btree-select, the Btree-join [4], the join index [16] or the path index [6]. We call the join between oid-tuple lists[2] the *Oid-join*.

We extend the Oid-intersection and Oid-join techniques to the spatial and non-spatial mixed query processing using B+-trees and R*-trees.

3. New Optimization Strategies for Mixed Queries

We have the following assumptions in this section:

(1) We consider only SEL(ECT) and JOIN operations among the SOA operators because only both the operators are able to have spatial or non-spatial predicates. In addition, throughout this paper, we sometimes attach "S_" or "N_" prefix to SEL and JOIN to distinguish spatial operators and non-spatial operators, which is only for an explanation.

(2) We consider only the R*-tree as a spatial indexing and the B+-tree as a non-spatial indexing.

(3) The separations of filter and refinement steps are possible only when the indexes are available for all of the attributes that the operations reference. Therefore, we consider only the *Rtree-select* [3] and the *Rtree-join* [5, 8] as the spatial filter algorithm. These need the random access using object identifiers at the refinement step. In this paper, this algorithm for the refinement step is called the *Obj-select*.

[1] This is also called the cascade of select rule.

[2] A simple oid list and an oid-pair list are also kinds of oid-tuple lists.

3.1. Early Separation of Filter and Refinement

As we mentioned in Section 1, the spatial query has been processed in two steps due to a large volume and high complexity of spatial data. However, this approach has been considered not in the query optimization phase but in the query execution phase. The state-of-the-art query optimizers did not separate the filter and refinement steps hidden in the algebraic operators from the optimization phase. They converted the filter and refinement steps together to one physical operator. However, when spatial subqueries and non-spatial subqueries are mixed and if spatial indexes exist on a class referenced by spatial subqueries, the separation of filter and refinement steps starting from the algebraic operator level can provide opportunities to generate more efficient execution plans to the optimizer. For example, suppose that the following mixed query which consists of a spatial predicate and a non-spatial predicate was issued by the user.

OQL 1 `select a from a in buildings where a.shape` *s_inside* `s_rectangle`(x_1, y_1, x_2, y_2) `and a.comp_date < ''80/01/01'';`

Equation (1) is an SOA-expression which is generated from the OQL parser for the above query.

$$\text{N\_SEL a.comp\_date} < \text{“80/01/01” (S\_SEL a.shape s\_inside s\_rectangle}(x_1, y_1, x_2, y_2)\ \text{(buildings : a))} \quad (1)$$

If the separation of filter and refinement is possible at the algebraic operator level of the query optimizer, i.e., if an R*-tree exists for the spatial attribute “shape” of the class “buildings”, the spatial select operation (S_SEL) in Equation (1) can be separated into the spatial select filter (SSF) and the spatial select refinement (SSR) operations. Equation (2) shows the separation.

$$\text{N\_SEL a.comp\_date} < \text{“80/01/01”} \left(\text{SSR a.shape s\_inside s\_rectangle}(x_1, y_1, x_2, y_2)\ \text{(SSF a.shape s\_inside s\_rectangle}(x_1, y_1, x_2, y_2)\ \text{(buildings : a))}\right) \quad (2)$$

Obviously, SSR is a SEL operation of the relational algebra because it is generated from a select operation (S_SEL). Therefore, the query in Equation (2) can be transformed to the query in Equation (3) which is in the order of “filter step – non-spatial operation – refinement step” by the select commutative rule of the relational algebra.

$$\text{SSR a.shape s\_inside s\_rectangle}(x_1, y_1, x_2, y_2) \left(\text{N\_SEL a.comp\_date} < \text{“80/01/01” (SSF a.shape s\_inside s\_rectangle}(x_1, y_1, x_2, y_2)\ \text{(buildings : a))}\right) \quad (3)$$

The processing of the original query in the order of Equation (3) can be more efficient than the order of “spatial operation – non-spatial operation” like Equation (1) or “non-spatial operation – spatial operation”. If the filter and refinement steps are not separated at the algebraic operator level, the query in Equation (3) cannot be generated, but only the execution plan like “spatial operation – non-spatial operation” or “non-spatial operation – spatial operation” can be generated.

The spatial join operation can also be separated into the spatial join filter (SJF) and the spatial join refinement (SJR). And, SJR, like SSR, is a SEL operation of the relational algebra because the join operation between two input classes has already been performed by SJF which is the filter step (Remind the following relational algebra rule: JOIN $\theta_1 \wedge \theta_2(R, S)$ = SEL θ_1(JOIN $\theta_2(R, S)$).). The following is an example query involving a spatial join operation.

OQL 2 `select a from a in buildings, b in roads where a.shape` *s_touch* `b.route and a.comp_date < ''80/01/01'';`

Equation (4) is an SOA-expression for the above query.

$$\text{N\_SEL a.comp\_date} < \text{“80/01/01” (S\_JOIN a.shape s\_touch b.route (roads : b))} \quad (4)$$

As in the case of Equation (2), if R*-trees exist for both the input classes “buildings” and “roads”, the spatial join operation (S_JOIN) in Equation (4) can be separated into SJF and SJR shown in Equation (5).

$$\text{N\_SEL a.comp\_date} < \text{“80/01/01”} \left(\text{SJR a.shape s\_touch b.route (SJF a.shape s\_touch b.route (roads : b))}\right) \quad (5)$$

Since SJR is a SEL operation of the relational algebra, we can apply the select commutative rule of relational algebra to the above query. Equation (3) is the resulting query.

$$\text{SJR a.shape s\_touch b.route} \left(\text{N\_SEL a.comp\_date} < \text{“80/01/01” (SJF a.shape s\_touch b.route (roads : b))}\right) \quad (6)$$

If the spatial join selectivity is low and the non-spatial select selectivity is middle to high, the query of Equation (3) can be more efficient than the query which are in the order of “spatial join – non-spatial select” or vice versa. However, as in the case of spatial select, if the filter and refinement steps are not separated at the algebraic operator level, the execution plan such as Equation (3) cannot be generated.

As we saw in the above two examples, separating a spatial operation into filter and refinement steps at the algebraic operator level enables the optimizer to generate more efficient execution plans in some cases. Therefore, the first optimization strategy for mixed queries is as follows:

Strategy 1 Separate spatial operations into filter step operations and refinement step operations at the algebraic operator level.

3.2. Combined Refinement

During the mixed query optimization, the refinement operation can be combined with other non-spatial operations to be processed in a unit. As we mentioned in the previous section, all refinement operations correspond to the SEL operation of the relational algebra. Therefore, due to the select-merge (cascade of select) rule of the relational algebra [15], the refinement operation can be combined with other non-spatial select operations to generate another SEL operation. For example, N_SEL and SSR in Equation (2) and Equation (3) can be combined and converted into a SEL operation. Likewise, N_SEL and SJR in Equation (5) and Equation (6) can also be combined into a SEL operation shown in Equation (7).

$$\text{SEL a.comp\_date} < \text{“80/01/01”} \wedge \text{a.shape s\_touch b.route}\ (\text{SJF a.shape s\_touch b.route}\ (\text{buildings} : \text{a}, \text{roads} : \text{b})) \tag{7}$$

Since all the refinement operations correspond to the SEL operation of the relational algebra, they can also be combined with other spatial refinement operations in addition to non-spatial select operations. We call the combining of non-spatial select operations and spatial refinement operations the *combined refinement*. The second strategy for the mixed query optimization is the combined refinement.

Strategy 2 (Combined Refinement) Combine the refinement steps of the spatial operations with non-spatial select operations or other spatial refinement operations by the select-merge rule.

3.3. Combined Filtering

As in the case of the combined refinement, the spatial filter operation can be combined with other non-spatial filter operations if the non-spatial operations can be evaluated by the indexes. This can be done by applying the *Oid-intersection* technique [9] and the *Oid-join* technique [4, 16, 6] to spatial and non-spatial mixed query processing.

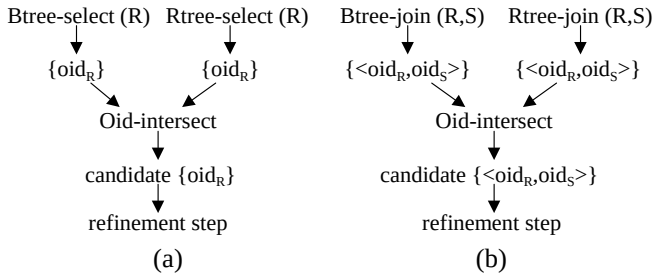

Figure 1. Oid-intersection between spatial and non-spatial operations

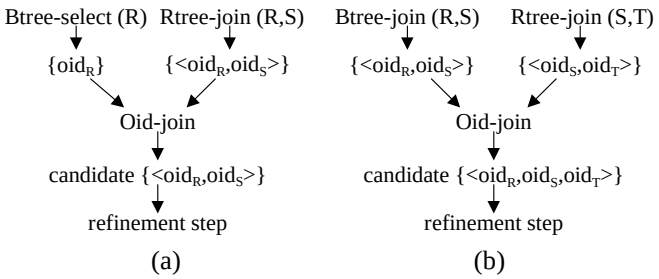

Figure 2. Oid-join between spatial and non-spatial operations

Figure 1 and Figure 2 show the Oid-intersection technique and the Oid-join technique, respectively, between the results of a typical spatial operation and a typical non-spatial operation. In the above figures, *Oid-intersect* and *Oid-join* are the INTERSECT operation and the NATURAL JOIN operation between oid-tuple collections, respectively. Since the index probing for the spatial operation can obtain the oid-tuple collection only for candidate objects, we append the refinement step to the original Oid-intersection or Oid-join technique to obtain the actual result.

If the B+-tree index exists for the attribute “comp_date” of the class “buildings” in Equation (3), the N_SEL and SSF operations can be combined by the Oid-intersection technique. Likewise, the N_SEL and SJF operations in Equation (6) can also be combined to generate the execution plan of Equation (8) by the Oid-join technique. The interesting fact in Equation (8) is that the non-spatial operation is separated into filter and refinement in order to apply the Oid-join technique. The separation of non-spatial operations will be mentioned in Section 4.

$$\text{Obj\_select a.shape s\_touch b.route} \wedge \text{a.comp\_date} < \text{“80/01/01”}\ \Big(\text{Oid\_join}\ (\text{Rtree\_join a.shape s\_touch b.route}\ (\text{buildings} : \text{a}, \text{roads} : \text{b}), \text{Btree\_select a.comp\_date} < \text{“80/01/01”}\ (\text{buildings} : \text{a}))\Big) \tag{8}$$

As in the case of the combined refinement, the spatial filter operations can be combined with other spatial filter operations in addition to non-spatial filter operations. This combining of non-spatial filter operations and spatial filter operations is called the *combined filtering*, which is the third strategy for the mixed query optimization.

Strategy 3 (Combined Filtering) Combine the spatial filter operations with non-spatial filter operations or other spatial filter operations using the Oid-intersection or Oid-join technique.

Since the intersect operation is a special case of the join

operation,[3] we will consider only the Oid-join from now on. Since the Oid-join is an operation between only oid-tuple collections, its cost may be much cheaper than the join between object-tuple collections. The combined filtering uses the spatial indexes and the non-spatial indexes as much as possible and does not generate the intermediate results except the oid-tuple collections. Therefore, we expect that Strategy 3 will have a considerable effect in the spatial and non-spatial mixed query processing.

We have suggested three optimization strategies for mixed queries. However, these strategies by themselves cannot always generate the most efficient plan. Therefore, we should check each strategy against a cost model for the input query. In the next section, we present a rule-based optimization technique for these optimization strategies.

4. Intermediate Spatial Object Algebra and Optimization Rules

4.1. Separation of Non-spatial Operators

In Section 3, we separated some algebraic operators of the SOA into those of filter and refinement steps. When B+-trees were already built on the classes which are referenced in the non-spatial predicates of the query, the processing of the non-spatial operation can be regarded as the two-step processing like that of the spatial operation. As the first step, we can obtain the object identifiers of the query result by accessing only the B+-trees. At the second step, the objects in the database are retrieved using the object identifiers which are acquired at the first step. We also call the first step the *filter step* and the second step the *refinement step* for the non-spatial query. Obtaining the object identifiers not for the candidates but for the exact result is different from the spatial filter and refinement. There is no extra CPU-time in the refinement step because the step only retrieves the real objects for the object identifiers which are acquired in the filter step. The reason why we separate a non-spatial operator into the filter and refinement steps is for the uniform treatment between spatial operators and non-spatial operators. In this paper, *Btree-select* and *Btree-join* [4] algorithms are used as non-spatial filter algorithms.

4.2. Intermediate Spatial Object Algebra (ISOA)

The operators which correspond to the filter step or the refinement step are actually in the intermediate form between the algebraic operators and the physical operators because the operators of the filter and refinement steps can be separated only when indexes, which are physical elements, exist. We call such an intermediate form of the SOA the Intermediate Spatial Object Algebra (ISOA). The ISOA has the following operators in addition to the SOA operators:

(1) filter step operators
SSF, SJF, NSF (Non-spatial Select Filter), NJF (Non-spatial Join Filter), OJ (Oid-Join)

(2) refinement step operators
SSR, SJR, NSR (Non-spatial Select Refinement), NJR (Non-spatial Join Refinement)

As in the case of the SOA, the spatial operators and non-spatial operators are not actually distinguished in the ISOA. Therefore, SSF and NSF are represented as SF (Select Filter), SSR and NSR as SR (Select Refinement), SJF and NJF as JF (Join Filter), and SJR and NJR as JR (Join Refinement). In addition, as we mentioned in Section 3, all the refinement operations correspond to the SEL operation of the relational algebra.

4.3. Optimization Rules for ISOA

In this section, we present optimization rules for Strategy 1 and Strategy 3 using the ISOA. As we mentioned in Section 3, the optimization rule for Strategy 2 is derived by the select-merge rule of the relational algebra. First, we present optimization rules for Strategy 1.

In the following rules, θ_R denote a spatial predicate or non-spatial predicate for the class R, and $\theta_{R,S}$ denote a spatial predicate or non-spatial predicate between the classes R and S. We will omitt the proofs for the following rules because they are obvious from the definitions of SF, JF, SR and JR.

Rule 1 (SEL θ_R (R)) = (SR θ_R (SF θ_R (R)))

Rule 2 (JOIN $\theta_{R,S}$ (R, S)) = (JR $\theta_{R,S}$ (JF $\theta_{R,S}$ (R, S)))

If the inputs of an SOA operator are base classes and the indexes exist on the classes, the separation of the operator into the filter and refinement steps is obvious by Rule 1 and Rule 2. If an operator has intermediate results as inputs, the filter and refinement steps of the operator cannot be directly separated because the intermediate result does not have an index. However, in the case that the intermediate results are collections of oid-tuples resulting from the previous filter step operators, and the indexes exist on the base classes which the operator references, the separation of the operator into the filter and refinement steps is possible with the Oid-join. In this case, the filter step of the operator is the index probing on the base classes, and the Oid-join is the join between the oid-tuple collection resulting from the filter step of the operator and the oid-tuple collection resulting

[3] If the types of the two input tuple collections to be joined are the same, the natural join between the collections becomes the intersect operation.

from the previous filter step operators. In this way, the combined filtering can be done by applying the separation of an operator which has oid-tuple collections as inputs and the Oid-join simultaneously.

The following rules are about the separation of the non-spatial or spatial operation and the Oid-join technique for complex mixed queries. In the following rules, E_R, E_S and $E_{R,S}$ denote oid-tuple collections whose tuple elements include the oid of the class R, the class S and both, respectively.

Rule 3 (SEL θ_R (E_R)) = (SR θ_R (OJ (E_R, SF θ_R (R))))

Rule 4 (SEL $\theta_{R,S}$ ($E_{R,S}$)) = (JR $\theta_{R,S}$ (OJ ($E_{R,S}$, JF $\theta_{R,S}$ (R, S))))

Rule 5 (JOIN $\theta_{R,S}$ (E_R, S)) = (JR $\theta_{R,S}$ (OJ (E_R, JF $\theta_{R,S}$ (R, S))))

Rule 6 (JOIN $\theta_{R,S}$ (R, E_S)) = (JR $\theta_{R,S}$ (OJ (JF $\theta_{R,S}$ (R, S), E_S)))

Rule 7 (JOIN $\theta_{R,S}$ (E_R, E_S)) = (JR $\theta_{R,S}$ (OJ (E_R, OJ (JF $\theta_{R,S}$ (R, S), E_S))))

Theorem 1 *The above rules are correct.*

Proof We will only prove Rule 7 which is most complicate. Other rules can be proved in similar manners. In the following proof, NJ denotes the NATURAL JOIN operator of relational algebra.

Since E_R is an oid-tuple collection resulting from the previous filter step operators, the number of the distinct objects for the class R in E_R is less than or equal to that of the base class. The same case is held between the class S and E_S. Therefore,

JOIN $\theta_{R,S}$ (E_R, E_S)

$=$ JOIN $\theta_{R,S}$ (NJ (E_R, R), NJ (S, E_S))

$=$ NJ (E_R, NJ (JOIN $\theta_{R,S}$ (R, S), E_S))

by join associativity

$=$ NJ (E_R, NJ (JR $\theta_{R,S}$ (JF $\theta_{R,S}$ (R, S)), E_S))

by Rule 2

$=$ JR $\theta_{R,S}$ (NJ (E_R, NJ (JF $\theta_{R,S}$ (R, S)), E_S))

by select join commutativity

$=$ JR $\theta_{R,S}$ (OJ (E_R, OJ (JF $\theta_{R,S}$ (R, S)), E_S))

since NJ between oid-tuple collections is OJ

□

In Equation (8), we showed the application of Rule 3. That is, Equation (8) is derived by applying Rule 3 to Equation (6) and converted to an execution plan. We will give another example to show the application of other rules to a complex query. Consider the following OQL query.

OQL 3 `select a from a in buildings, b in roads, c in districts where a.shape` *s_touch* `b.route and a.shape` *s_covered_by* `c.boundary and a.comp_date < ''80/01/01'' and c.boundary` *s_intersect* `s_rectangle`(x_1, y_1, x_2, y_2)`;`

Let $\theta_1, \theta_2, \theta_3$ and θ_4 be `a.shape` *s_touch* `b.route`, `a.shape` *s_covered_by* `c.boundary`, `a.comp_date < ''80/01/01''` and `c.boundary` *s_intersect* `s_rectangle`(x_1, y_1, x_2, y_2), respectively. Equation (9) is an SOA-expression for OQL 3.

$$\text{S\_JOIN } \theta_2 \left(\text{S\_JOIN } \theta_1 \text{ (N\_SEL } \theta_3 \text{ (a), b), S\_SEL } \theta_4 \text{ (c)}\right) \quad (9)$$

By applying Rule 1, the select-join commutative rule and the select-merge rule of the relational algebra to Equation (9),

$$\text{SEL } \theta_3 \wedge \theta_4 \left(\text{S\_JOIN } \theta_2 \left(\text{S\_JOIN } \theta_1 \text{ (NSF } \theta_3 \text{ (a), b), SSF } \theta_4 \text{ (c)}\right)\right) \quad (10)$$

By Rule 5, the select-join commutative rule and the select-merge rule,

$$\text{SEL } \theta_3 \wedge \theta_4 \wedge \theta_1 \left(\text{S\_JOIN } \theta_2 \left(\text{OJ (NSF } \theta_3 \text{ (a), SJF } \theta_1 \text{ (a, b)), SSF } \theta_4 \text{ (c)}\right)\right) \quad (11)$$

By Rule 7, the select-join commutative rule and the select-merge rule,

$$\text{SEL } \theta_3 \wedge \theta_4 \wedge \theta_1 \wedge \theta_2 \left(\text{OJ } \left(\text{OJ (NSF } \theta_3 \text{ (a), SJF } \theta_1 \text{ (a, b)), OJ (SJF } \theta_2 \text{ (a, c), SSF } \theta_4 \text{ (c))}\right)\right) \quad (12)$$

Since OJ is a natural join between Oid-tuple collections, the commutative rule and the associative rule can be derived.

Rule 8 (OJ (E_1, E_2)) = (OJ (E_2, E_1))

Rule 9 (OJ (OJ (E_1, E_2), E_3)) = (OJ (E_1, OJ (E_2, E_3)))

5. Implementation and Experiments

We implemented the ESFAR optimization technique proposed in Section 3 and Section 4 using the Volcano optimizer generator (VOG) [7]. Using the implementation, we measured the effect of our optimizer for several types of mixed queries. Three types of queries are used: "non-spatial select – spatial select" which is an OQL 1 type; "non-spatial select – spatial join" which is an OQL 2 type; and "3-way spatial join" as a sequnece of 2-way joins. We measured the expected response time for the above three types of queries using three optimization methods: (1) a

traditional method which does not separate filter and refinement steps, denoted by TRA; (2) the method using only Strategy 1, denoted by SEP; and (3) the method including Strategy 2 and Strategy 3 in addition to Strategy 1, denoted by CFR. The major assumptions for the experiments are as follows:

(1) Both spatial and non-spatial data are uniformly distributed, and there is no buffering and clustering.

(2) We consider only I/O time for non-spatial operations, I/O time for spatial filter operations, and both CPU and I/O time for spatial refinement operations. The CPU time for spatial refinement operations is only for geometric computation time since other components are negligible.

The parameters used in the experiments are shown in Table 1. We assume that the parameters for two input classes in 2-way spatial joins (for example, $||R||$ and $||S||$, s_R and s_S) are the same. We assume that the average number of points per spatial object is 20. This assumption will be also applied in 3-way joins. The physical algorithms and a cost model applied in our experiments are summarized in [13].

Table 1. Parameters used in experiments

parameters	descriptions	values
$\|\|R\|\|$	Number of objects in class R^*	100,000
D^2	Area of total space	$10^5 * 10^5$
s_R	Size of non-spatial parts of an object in R	200 bytes
Page_Size	Page size	4096 bytes
Point_Size	Size of a point	8 bytes
DA	Disk access time	10 ms
$T_{edge-rect}$	CPU time for edge-rectangle test	40 μs
$T_{edge-edge}$	CPU time for edge-edge test	20 μs
hit_S	Hit ratio for spatial selection	90 %
hit_J	Hit ratio for spatial join	70 %

$^*R$ contains both non-spatial parts and spatial parts

For the query of "non-spatial select – spatial select" type, we measured the expected response time with the varying non-spatial selectivity ($S_{\theta_{nsp}}$) and spatial selectivity ($S_{\theta_{sp}}$) such as a high selectivity (1/2), a middle selectivity (1/16) and a low selectivity (1/128). Table 2 shows the result. In fact, SEP and CFR always generate more efficient execution plans than TRA because the set of optimization rules associated with SEP or CFR is a super set of that with TRA. Therefore, an important measure in these experiments is the expected performance rate which is defined as follows:

$$\text{expected performance rate} = \text{expected response time of TRA} \Big/ \text{expected response time of SEP or CFR}$$

As shown in Table 2, SEP and CFR generate more efficient execution plans than TRA for the most cases. The performance rate of SEP compared with TRA is maximized in the high selectivity and that of CFR is maximized in the middle selectivity.

Table 2. Expected performance rate for the type of OQL 1

$S_{\theta_{nsp}}$	$S_{\theta_{sp}}$	TRA/SEP	TRA/CFR
1/2		1.21	1.39
1/16	1/4	1.20	2.47
1/128		1.16	1.16
	1/2	1.34	1.35
1/4	1/16	1.14	2.53
	1/128	1.11	1.13

Table 3. Expected performance rate for the type of OQL 2

$S_{\theta_{nsp}}$	MBR size	TRA/SEP	TRA/CFR
1/2		1.18	1.97
1/16	100*200	1.03	1.49
1/128		1.03	1.03
	25*25	1.11	1.61
1/4	100*100	1.20	3.38
	400*400	1.01	1.01

Next, we conducted an experiment for "non-spatial select – spatial join" with the varying non-spatial selectivity and spatial selectivity. The selectivity of a spatial join can be represented by an average MBR size (For details, refer to [13]). Table 3 shows the result. In case of the varying non-spatial selectivity, SEP is beneficial only in case of the high selectivity and CFR is beneficial in cases of the high selectivity and the middle selectivity. In case of the low selectivity, we can predict that processing the query in the order of "non-spatial select – spatial join" is more efficient than that of "spatial join filter – non-spatial select – spatial join refinement" or "Btree-select – Rtree-join – Oid-join – combined refinement." In case of the varying spatial selectivity, the performance rates of SEP and CFR compared with TRA are maximized in the middle selectivity.

Last, we measured the expected response time for a 3-way spatial join (as a sequence of 2-way joins) with a varying average MBR size (Table 4). In this experiment, Strategy 1 does not contribute to reduce the response time for itself because both of the joins are spatial joins. Therefore, the experimental result only for CFR is shown in Table 4. TRA generates an execution plan which consists of an Rtree-join with the refinement, and an indexed nested loop join (INLJ), or a plan consisting of two indexed nested

Table 4. Expected performance rate for the type of 3-way spatial join

MBR size	TRA/CFR
25*25	1.85
100*100	3.07
400*400	1.00

loop joins. On the other hand, CFR generates an execution plan which consists of the Oid-join between two Rtree-joins with the combined refinement in addition to those by TRA. The performance gains in the small MBR size and the middle MBR size are high. This is because the combined filtering in a multi-way join uses all available indexes and does not generate intermediate results except oid-tuple collections. However, when the average MBR size is large, our optimization technique is not beneficial. In this case, the probability of overlapping between MBR's becomes high, which results in a large number of MBR pairs in a result of the Rtree-join, and therefore produces many oid-tuples. The ESFAR optimization technique in the multi-way spatial join is beneficial when the number of MBR-pairs resulting from the Rtree-join is less than the average number of input MBR's. This corresponds to the case that an average MBR size is less than about 150*200. According to some experimental results using real data such as the TIGER file in the literature [5, 8], the size of a join result is less than the average size of input data in most cases. Therefore, the ESFAR optimization technique is considered to be beneficial in a real environment.

6. Conclusions

In this paper, we proposed a query optimization technique which took the characteristics of spatial databases into account. The main idea is to start the two-step processing of a spatial query, which has been applied only in the query execution phase, from the query optimization phase. We showed that filter and refinement operations could be separated at the algebraic operator level of the query optimization, then the separated filter and refinement operators could be combined with other non-spatial operators or spatial operators at the same level. We called this optimization technique ESFAR. To implement ESFAR, we defined ISOA and optimization rules using it.

We implemented the ESFAR optimization technique using VOG. We showed that our ESFAR optimization technique generated more efficient execution plans than those by traditional optimization techniques. In addition, the ESFAR optimization technique improved the performance of the query processing for a reasonable range of selectivities.

References

[1] W. Aref and H. Samet, "Optimization Strategies for Spatial Query Processing," Proc. of VLDB, 81-90, 1991.

[2] L. Becker and R. H. Güting, "Rule-based Optimization and Query Processing in an Extensible Geometric Database System," ACM Transactions on Database Systems, Vol. 17, No. 2, 247-303, 1992.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," Proc. of ACM SIGMOD, 322-331, 1990.

[4] M. Blasgen and K. Eswaran, "Storage and Access in Relational Databases," IBM Systems Journal, Vol. 16, No. 4, 363-377, 1977.

[5] T. Brinkhoff, H.-P. Kriegel and B. Seeger, "Efficient Processing of Spatial Joins Using R-trees," Proc. of ACM SIGMOD, 237-246, 1993.

[6] W.-S. Cho, K.-Y. Whang, S.-S. Lee and Y.-I. Yoon, Query Optimization Techniques Utilizing Path Indexes in Object-Oriented Database Systems," Proc. of DASFAA, 21-29, 1997.

[7] G. Graefe and W. J. McKenna, "The Volcano Optimizer Generator: Extensibility and Efficient Search," Proc. of IEEE ICDE, 209-218, 1993.

[8] Y.-W. Huang, N. Jing and E. A. Rundensteiner, "Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations," Proc. of VLDB, 396-405, 1997.

[9] C. Mohan, D. Haderlr, Y. Wang and J. Cheng, "Single Table Access Using Multiple Indexes: Optimization, Execution, and Concurrency Control Techniques," Proc. of EDBT, 29-43, 1990.

[10] B. C. Ooi, R. Sacks-Davis and K. J. McDonell, "Extending a DBMS for Geographic Applications," Proc. of IEEE ICDE, 590-597, 1989.

[11] J. A. Orenstein, "Spatial Query Processing in an Object-Oriented Database System," Proc. of ACM SIGMOD, 326-336, 1986.

[12] H.-H. Park, N.-H. Hong, C.-W. Park and C.-W. Chung, "Spatial Object Algebra and Query Language in OMEGA," KAIST, Technical Report, CS/TR-97-118, 1997.

[13] H.-H. Park, C.-G. Lee, Y.-J. Lee and C.-W. Chung, "Separation of Filter and Refinement Steps in Spatial Query Optimization," KAIST, Technical Report, CS/TR-98-122, 1998. See also: http://islab.kaist.ac.kr/~hhpark/eng_tr_sfro.ps

[14] H. Samet and W. G. Aref, "Spatial Data Models and Query Processing," Modern Database Systems, ACM Press, 338-360, 1995.

[15] A. Silberschatz, H. F. Korth and S. Sudarshan, Database System Concepts, McGrawHill, 3rd edition, 1997.

[16] P. Valduriez, "Join Indices," ACM Transactions on Database Systems, Vol. 12, No. 2, 218-246, 1987.