SPATIAL QUER Y OPTIMIZATION UTILIZING EARLY SEPARATED FILTER AND REFINEMENT STRATEGY†

HO-HYUN PARK1, YONG-JU LEE2 and CHIN-WAN CHUNG1

1Department of Computer Science, KAIST, South Korea
2Department of Computer Engineering, Sangju National University, South Korea

(Received 20 September 1998; in final revised form 2 February 2000)

Abstract — Due to the high complexity and large volume of spatial data, a spatial query is usually processed in two steps, called the filter step and the refinement step. However, the two-step processing of the spatial query has been considered locally in one spatial predicate evaluation at the query execution level. This paper presents query optimization strategies which exploit the two-step processing of a spatial query at the query optimization level. The first strategy involves 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 if they were already separated, 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 an algebra, which is called the Intermediate Spatial Object Algebra (ISOA), and optimization rules for ESFAR. Through experiments using real data, we compare the ESFAR optimization technique with a traditional optimization technique which does not separate filter and refinement steps from the query optimization phase. The experimental results show that the ESFAR optimization technique generates more efficient query execution plans than the traditional one in many cases. ©2000 Elsevier Science Ltd. All rights reserved.

Key words: Spatial Databases, Spatial Query Optimization, Filter, Refinement, ESFAR

1. INTRODUCTION

Due to the application areas such as geographic information systems (GISs), computer aided design (CAD) and multimedia processing, user requirements for spatial data handling in database systems are steadily increasing. However, conventional alpha-numeric database management systems could not properly cope with the increasing applications. This caused a lot of research on spatial database management systems such as GEOQL [22], GRAL [4], SAND [2] and Paradise [17].

One of the research areas is the query processing. The processing cost of a spatial query is very expensive because spatial data is more complex and larger than alpha-numeric data. Therefore, the spatial query has been processed mostly in two steps [8, 23]. At the first step which is called the filter step, complex spatial objects are approximated to simpler objects. Then, the query is processed against the approximated objects. The set of objects which passed the filter step is not the exact result for the original query but the set of candidate objects which is a super set of the result. These candidate objects are further processed using geometric algorithms to obtain an exact result at the next step which is called the refinement step. The approximation methods which are used at the filter step include the minimum bounding rectangle (MBR) [5, 15], Z-elements [23] and 5-corner [8]. These approximations are mostly stored in spatial indexes such as the R-tree [15], R*-tree [5] and Z-elements using the B+-tree [23].

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 predicates and non-spatial predicates. However, the two-step processing of the spatial query has been considered locally in one spatial predicate evaluation at the query execution level. For example, a spatial join can be processed in a multi-step algorithm as presented in [8]. However, most of the existing spatial query optimizers regarded the multi-step join algorithm as one algebraic operator when they determine the execution order of algebraic operators including the join operator. If each step of the multi-step join

†Recommended by Yannis Ioannidis
algorithm is regarded as an individual algebraic operator at the query optimization level, more
diverse and efficient query execution plans such as the following can be generated: 1) non-spatial
operators can be interleaved between the filter step operator and the refinement step operator by
algebraic rules, 2) several refinement step operators can be combined into one algebraic operator, and
3) several filter step operators can also be combined.

Several spatial database systems have addressed the optimization problem for the spatial and
non-spatial mixed query in the literature [32]. GEOQL’s optimizer decomposes a spatial query into
spatial subqueries and non-spatial subqueries [22]. The decomposed subqueries are optimized separately
and participate in an order that minimizes the overall query cost. The non-spatial subqueries
are processed by the SQL backend and the spatial subqueries by the spatial processor. Because of
the decomposition, this optimization technique may prohibit an arbitrary ordering between spatial
operators and non-spatial operators and decrease the optimization quality. GRAL [4] performs a
rule-based optimization using an algebra which is called the geo-relational algebra. GRAL’s optimi-
zer uses a pre-defined partial order between some algebraic operators to find a good execution
order in a heuristic manner. SAND’s optimizer [2] provides an equal opportunity for both spatial
and non-spatial data and uses several optimization strategies to make an efficient ordering and
merging for spatial and non-spatial operations. Paradise uses basic relational operators and opti-
mization techniques for both spatial and non-spatial operations [31]. Paradise’s query optimizer
is written by an optimizer generator called OPT++ [17]. However, none of the above optimizers
provides the filter and refinement steps of spatial operators separately 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 exploit the two-step processing of a
spatial query at the query optimization level. The first strategy involves 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 predicate and
a non-spatial predicate, the processing order 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”. For the early separation of the filter and refinement steps,
we introduce algebraic operators which denote filter steps or refinement steps of some operators.
As the second strategy, several refinement operations can be combined in processing a complex
query if they were already separated, and as the third strategy several filter operations can also be
combined. We use some relational algebra optimization rules for combining refinement steps [33],
and the Oid-intersection technique [21] and the Oid-join technique [7, 12, 35] for combining filter
steps. We call the optimization technique utilizing these strategies the Early Separated Filter And
Refinement (ESFAR).

We defined the Spatial Object Algebra (SOA), which extends a normal object algebra to process
spatial predicates as well, in our previous research [27], and use SOA 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 and several optimization rules using it.

To measure the performance of the ESFAR optimization technique, we implemented the opti-
mizer using the Volcano optimizer generator (VOG) [14] and execution algorithms for ISOA. Since
the simulation results for the optimizer using a cost model were presented in the early ver-
sion of this paper [28, 29], we concentrate on real execution for the ESFAR strategies. Through
experiments using the TIGER data [34], we compare the ESFAR optimization technique with a
traditional optimization technique which does not provide the filter and refinement steps as in-
dividual operators. The experimental results show that in many cases 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 previous work
which is regarded as the background for our work. In Section 3, we explain ISOA which is an
algebraic framework for the ESFAR optimization technique. The ESFAR optimization strategies
and its optimization rules using ISOA are explained in Section 4. The implementation and ex-
experimental results are shown in Section 5. In Section 6, we conclude this paper and suggest some future studies.

2. BACKGROUND


2.1.1. Algorithms Using Indexes

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

As spatial selection algorithms, there are the window query [5, 15] and the topological queries with spatial constants [25], etc. In this paper, we call the algorithms for such queries the R-tree-select. In the spatial join area, when R*-trees exist on both join inputs, a join algorithm which synchronously traverses both R*-trees by the depth-first search was proposed [9]. 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 [16]. In this paper, we call both of the join algorithms the R-tree-join.

The B-tree-select is an algorithm for a range query for the B+-tree. In the non-spatial join, if B+-trees exist on both join inputs, the B-tree-join similar to the R-tree-join is possible. The B-tree-join synchronously traverses only the leaf nodes of both B+-trees by the merge-join technique. The B-tree-join technique was already used in [7], and was especially efficient when both indexes were clustered.

2.1.2. Oid-Intersection

When a query refers to multiple indexes in a single class, the index intersection technique and the index union technique can be applied [21]. The index intersection technique was used in the case of a conjunction of simple predicates and the index union technique in the case of a disjunctive predicate. We consider only conjunctive predicates in this paper. The main idea of the index intersection technique focuses on the Oid-intersection between the object identifier (oid) lists resulting from each index probing. We extend the Oid-intersection technique to the spatial and non-spatial mixed query processing using B+-trees and R*-trees.

2.1.3. Oid-Join

When a query has both select and join operations and indexes exist on all join attributes as well as selection attributes, the query can be evaluated by using indexes and oids [7, 35]. This can be performed by the natural join between the oid list from the B-tree-select and the oid-pair list from the B-tree-join [7]. The oid-pair list can also be produced from the join index [35].

The index intersection technique in [21] was extended in [12] to the path index of object-oriented databases. When a query can be processed by using both the path index and the simple index, the oid list from the simple index and the oid-tuple list from the path index can be joined to obtain the oid-tuple list for the query result. This method was called the generalized index intersection or index join.

We call both of the above join algorithms between oid-tuple lists\(^\dagger\) the Oid-join and extend the Oid-join technique to the spatial and non-spatial mixed query processing using B+-trees and R*-trees. The Oid-intersection and the Oid-join will be extended even to algorithms of non-indexed inputs in this paper.

\(^\dagger\)A simple oid list and an oid-pair list are also kinds of oid-tuple lists.
interface Building
( extent buildings)
{
    attribute string name;
    attribute integer year;
    attribute string usage;
    attribute S\_G-polygon shape;
};

interface Road
( extent roads)
{
    attribute string name;
    attribute string type;
    attribute integer year;
    attribute S\_Line route;
};

Fig. 1: Example of Class Definitions which Have Spatial Attributes

2.1.4. Algorithms without Using Indexes

When B+-trees or R*-trees do not exist, we must scan the whole space of the relation for a selection operation. We call such an algorithm the \textit{Seq-select}. Non-spatial join algorithms using no index involve the \textit{Sort-Merge join} and the \textit{Hash join}. For the case of one index, the \textit{Indexed-Nested-Loop join} algorithm exists. Likewise, for the spatial join using no index or one index, there are the \textit{Partition-Based-Spatial-Merge (PBSM) join} [30], the \textit{Spatial-Hash join} [19] and the \textit{Seeded-Tree join} [18], etc.

The algorithms using indexes such as the \textit{B-tree-select}, the \textit{B-tree-join}, the \textit{R-tree-select} and the \textit{R-tree-join}, and the filter step of some spatial operations such as the \textit{PBSM join} and the \textit{Seeded-Tree join} produce an oid list or oid-pair list as the result. The \textit{Obj-select} and the \textit{Obj-pair-select} [35] are the algorithms for scheduling the oid list and the oid-pair list, respectively, and for the refinement step of the spatial operations.

2.2. OMEGA

2.2.1. Data Model

OMEGA is a spatial object-oriented database management system which is currently under development at the Korea Advanced Institute of Science and Technology. OMEGA complies with the object model of the ODMG standard [11] and supports ODL (Object Definition Language), OQL (Object Query Language) and C++ OML (Object Manipulation Language). In a previous study, we added the spatial data types and operations to the ODMG object model [27]. The primitive spatial data types are based on the definition of spatial objects in the Spatial Data Transfer Specification (SDTS) [1] and classified as S\_Point, S\_Line and S\_G-polygon by the dimension. Figure 1 shows an example of class definitions which have spatial attributes using the ODMG 2.0 ODL [11].

OMEGA uses the Shore storage system [10] which provides the B+-tree and the R*-tree to support a fast access for objects and to speed up the filter step operation of spatial queries.

2.2.2. Spatial Object Algebra (SOA)

The \textit{Spatial Object Algebra} (SOA) is an algebraic specification for the spatial and non-spatial mixed query processing in OMEGA. It is an extended version of the object algebra in typical object-oriented databases to process spatial predicates as well as non-spatial predicates [27].

The operators in SOA are classified into algebraic operators and predicate operators. The algebraic operators are similar to the logical operators in Open OODB [6], MOOD [13], Paradise [17, 31], etc. They process inputs and outputs with a collection as a unit. The algebraic operators consist of the following:

1. relational algebra operators such as \texttt{SELECT}, \texttt{PROJECT}, \texttt{JOIN}, \texttt{UNION}, and \texttt{DIFFERENCE},

2. NEST and UNNEST operators for set attributes in the extended relational model,
3. MAT operator for path expressions in the object-oriented model, and

4. conversion operators such as ASSET, ASBAG, and ASLIST.

The predicate operators consist of \(<\), \(\leq\), \(=\), etc. which are non-spatial predicates and \(\text{s\_disjoint}\), \(\text{s\_equal}\), \(\text{s\_overlap}\), etc. which are spatial predicates. Among the SOA operators, we consider only SELECT and JOIN in this paper because these operators are most general among spatial operator.

### 3. INTERMEDIATE SPATIAL OBJECT ALGEBRA (ISOA)

Before describing the ESFAR optimization strategy, we define a new algebra which is called the Intermediate Spatial Object Algebra (ISOA). It is an extended version of SOA to support the filter and refinement strategy at the algebraic operator level of the query optimizer. In ISOA, some SOA operators can be separated into the filter and refinement step operators at the algebraic operator level. The filter and refinement step operators are actually in an intermediate form between SOA and the physical execution plan because the separations are only for the purpose of optimization.

**Definition 1** The filter and refinement steps are defined as follows in ISOA.

1. The *filter step* of the spatial or non-spatial operator produces the object identifiers for the super set \(^1\) of the exact result of the spatial or non-spatial predicate. The members of the super set of the exact result are called the *candidate objects*.

2. The *refinement step* of the spatial or non-spatial operator fetches the candidate objects using the object identifiers which were obtained from the filter step, and evaluates the spatial or non-spatial predicate to produce the exact result of the predicate.

Definition 1 says non-spatial operators can also be separated into the filter and refinement steps like spatial operators. This ensures the uniform treatment between spatial operators and non-spatial operators. In contrast to the spatial filter, the non-spatial filter produces the object identifiers of the exact result, i.e., candidate objects are the same as the exact result. Thus, the refinement step of the non-spatial operators only retrieves the real objects for the object identifiers which were obtained from the filter step.

#### 3.1. Operators

In OMEGA, the *object* in the database has the *value* and its *identifier*. Therefore, an object is represented by a pair (identifier, value). The terminology and concept of the object and the value are the same as those of \(O_2\) [3].

**Definition 2** ISOA is an algebra \((S, \Sigma)\), where \(S\) is the set of operands which are collection of tuples and \(\Sigma\) is the set of operators which will be defined subsequently.

In Definition 2, the collection indicates the ODMG collection which can be a set, a bag, a list or an array [11]. We only consider a set among the ODMG collections in this paper. A collection is denoted by \{\cdots\}. In OMEGA, the operand of a query processing is the collection of tuples. In ISOA, two kinds of tuple types exist. The first is the *object-tuple* whose elements consist of the objects. The second is the *object identifier (oid) tuple* whose elements consist of only object identifiers. In SOA, there was only the object-tuple type. A tuple is denoted by \{\cdots\}. If the number of tuple elements is one, the notation \('\langle\cdots\rangle'\) may be omitted. Table 1 presents some notations which will be used in this paper.

\(^1\text{This is not always the proper super set.}\)
The following is the definition of the geometric computation operator in ISOA. It represents the predicate evaluation for the spatial select or spatial join operator using a geometric computation method.

1. Geometric Computation (GC: denoted by $G_{\theta_{sp}}$):

   $$G_{\theta_{sp}}(E) = \{ x | \forall x \in E, \theta_{sp}(x) \}$$
As we mentioned before, some spatial operators in SOA can be separated into the filter and refinement operators in ISOA. SELECT and JOIN are such operators.

**Definition 5** The following are the definitions of the ISOA spatial operators. In the following definitions, \( \hat{a} \) and \( \hat{b} \), respectively, indicate approximations of spatial objects \( a \) and \( b \) such as MBRs, and \( \theta_{sp}^f \) indicates the filter step predicate, for the actual predicate \( \theta_{sp} \), which will be applied to the approximations.

1. **Spatial Select Filter (SSF; denoted by \( \sigma_{\theta_{sp}}^f \)):** SSF applies a filter step predicate to approximated values of objects. The result type of SSF is an oid-tuple collection.
   \[
   \sigma_{\theta_{sp}}^f(R) = \{ i(a) \mid \forall a \in R, \theta_{sp}^f(\hat{a}) \} 
   \]

2. **Spatial Select Refinement (SSR; denoted by \( \sigma_{\theta_{sp}}^r \)):** SSR fetches values using an oid-tuple collection and evaluates the actual spatial selection predicate using a geometric computation method.
   \[
   \sigma_{\theta_{sp}}^r(F) = G_{\theta_{sp}}(\nu(F))
   \]
   The filter and refinement steps operators for the spatial join are similarly defined as follows:

3. **Spatial Join Filter (SJF; denoted by \( \Join_{\theta_{sp}}^f \)):** \( R \Join_{\theta_{sp}}^f S = \{ (i(a),i(b)) \mid \forall a \in R, \forall b \in S, \theta_{sp}^f(\hat{a},\hat{b}) \} \)

4. **Spatial Join Refinement (SJR; denoted by \( \Join_{\theta_{sp}}^r \)):** \( \Join_{\theta_{sp}}^r(F) = G_{\theta_{sp}}(\nu(F)) \)

The filter step predicate can be different from the actual predicate [25]. The following are some examples of the filter step predicates for topological relationships between polygons [25]:
\[
\begin{align*}
\text{covered}_l & = \text{covered}_l \lor \text{inside}_l \lor \text{equal}_l \\
\text{overlap}_l & = \text{overlap} \lor \text{covered}_l \lor \text{inside} \lor \text{equal} \lor \text{covers} \lor \text{contains} \\
\text{inside}_l & = \text{inside}
\end{align*}
\]
   The first equation means that if an actual predicate \( \text{covered}_l \) is applied for the filter step, the three predicates on the right hand side of ‘\( \lor \)’ must be applied. The meanings of the other two equations are similarly explained. For simplicity, in the sequel, we will regard the \( \text{intersects} \) (which indicates “not disjoint”) as the filter step predicate for all topological relationships.

The filter and refinement steps operators for non-spatial SELECT and JOIN such as Non-spatial Select Filter (NSF), Non-spatial Select Refinement (NSR), Non-spatial Join Filter (NJF) and Non-spatial Join Refinement (NJR) can be similarly defined. In non-spatial operators, the filter step predicate is the same as the actual predicate (i.e., \( \theta_{nsp}^f = \theta_{nsp} \)). Therefore, the geometric computation \( G_{\theta_{nsp}} \) in the refinement step is not needed.

Some SOA operators can also be defined between the oid-tuple collections. INTERSECT and NATURAL JOIN are such operators.

**Definition 6** The following are the definitions of the ISOA operators between oid-tuple collections.

1. **Oid-intersect (denoted by \( F_1 \cap F_2 \))** is the intersection between the oid-tuple collections.
2. **Oid-join (denoted by \( F_1 \Join F_2 \))** is the natural join between the oid-tuple collections.

In addition to the above operators, all SOA operators between object-tuple collections are included in ISOA operators. The oid fields of the object-tuple collection do not participate in any SOA operators unless they are explicitly specified. For example, the intersect operation between object-tuple collections is a value-based intersection, and the join attributes in the natural join between object-tuple collections consist of only attributes in value fields of the object-tuple. From Definition 4 and Definition 5, we observed the following fact.

**Observation 1** \( G_{\theta_{sp}} \) corresponds to the SELECT operator of the relational algebra.
4. ESFAR OPTIMIZATION STRATEGY AND OPTIMIZATION RULES

We have the following assumptions in this section:

1. We consider only SELECT and JOIN among the SOA operators because those operators are the most general among spatial operators.

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

4.1. Early Separation of Filter and Refinement

As we mentioned in Section 1, the spatial query has been processed in two steps due to the 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 predicates and non-spatial predicates are mixed, 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.

By Definition 5, the separation of the SELECT and JOIN operators into filter and refinement steps is obvious. Therefore, we have the following rules for the early separation of filter and refinement steps:

Rule 1 \( \sigma_{\theta_{\text{sp}}} (E) = \sigma_{\theta_{\text{sp}}} (\sigma_{\text{sp}}^f (E)) \) = \( \nu(\sigma_{\text{sp}}^f (E)) \)

Rule 2 \( \sigma_{\theta_{\text{sp}}} (E) = \sigma_{\text{sp}}^r (\sigma_{\text{sp}}^f (E)) = \nu(\sigma_{\text{sp}}^f (E)) \)

Rule 3 \( E_1 \Join_{\theta_{\text{sp}}} E_2 = \Join_{\theta_{\text{sp}}} (E_1 \Join_{\text{sp}}^f E_2) = \nu(E_1 \Join_{\text{sp}}^f E_2) \)

Rule 4 \( E_1 \Join_{\theta_{\text{sp}}} E_2 = \Join_{\theta_{\text{sp}}} (E_1 \Join_{\text{sp}}^f E_2) = \nu(E_1 \Join_{\text{sp}}^f E_2) \)

By Observation 1, all relational algebra rules related to the SELECT operator can be applied to \( \sigma_{\theta_{\text{sp}}} \). The following are several rules related to \( \sigma_{\theta_{\text{sp}}} \):

Rule 5 \( \sigma_{\theta_{\text{sp}}} (\sigma_{\theta_{\text{sp}}} (E)) = \sigma_{\theta_{\text{sp}}} (\sigma_{\theta_{\text{sp}}} (E)) \)

Rule 6 \( \sigma_{\theta_{\text{sp}}} (E_1) \Join_{\theta_{\text{sp}}} E_2 = \sigma_{\theta_{\text{sp}}} (E_1) \Join_{\theta_{\text{sp}}} E_2 \)

Rule 7 \( \sigma_{\theta_{\text{sp}}} (\sigma_{\theta_{\text{sp}}} (E)) = \sigma_{\theta_{\text{sp}}} (\sigma_{\theta_{\text{sp}}} (E)) \)

Rule 8 \( \sigma_{\theta_{\text{sp}}} (\sigma_{\theta_{\text{sp}}} (E)) = \sigma_{\theta_{\text{sp}}} (\sigma_{\theta_{\text{sp}}} (E)) \)

Rule 9 \( \sigma_{\theta_{\text{sp}}} (E_1 \Join_{\theta_{\text{sp}}} E_2) = E_1 \Join_{\theta_{\text{sp}}} E_2 \)

Rule 10 \( \sigma_{\theta_{\text{sp}}} (\sigma_{\theta_{\text{sp}}} (E)) = \sigma_{\theta_{\text{sp}}} (\sigma_{\theta_{\text{sp}}} (E)) \)

The following is an example of the separation of the spatial select operator.
Example 2 Consider an OQL query referencing classes in Figure 1:

OQL 1 select a, b from a in buildings, b in roads where a.shape s_covered_by s_polygon(x_1, y_1, ..., x_n, y_n) and a.year = b.year;

Equation (1) is an SOA-expression for OQL 1.

\[
\left( \sigma_{a.shape \ s\_covered\_by \ s\_polygon(x_1, y_1, ..., x_n, y_n)} (\text{buildings : a}) \right) \bowtie_{a.year=b.year} (\text{roads : b}) \tag{1}
\]

The spatial select operator \(\sigma_{s\_covered\_by}\) in Equation (1) can be separated into the spatial select filter and the spatial select refinement operators by Rule 2. Equation (2) shows the separation.

\[
\left( G_{a.shape \ s\_covered\_by \ s\_polygon(x_1, y_1, ..., x_n, y_n)} \left( \nu \left( \sigma_{a.shape \ s\_intersect \ s\_polygon(x_1, y_1, ..., x_n, y_n)} (\text{buildings : a}) \right) \right) \right) \bowtie_{a.year=b.year} (\text{roads : b}) \tag{2}
\]

By Observation 1, \(G\) corresponds to the SELECT operator of the relational algebra. Therefore, Equation (2) can be transformed to Equation (3) which is in the order of “filter step - non-spatial operation - refinement step” by Rule 6.

\[
G_{a.shape \ s\_covered\_by \ s\_polygon(x_1, y_1, ..., x_n, y_n)} \left( \nu \left( \sigma_{a.shape \ s\_intersect \ s\_polygon(x_1, y_1, ..., x_n, y_n)} (\text{buildings : a}) \right) \right) \bowtie_{a.year=b.year} (\text{roads : b}) \tag{3}
\]

Sometimes, the processing of the original query in the order of Equation (3) can be more efficient than the order of “spatial select - non-spatial join” (Equation (1)) or “non-spatial join - spatial select”. However, if the filter and refinement steps are not separated at the algebraic operator level, an expression such as Equation (3) cannot be generated.

As we saw in the above example, 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 (Early Separation of Filter and Refinement) Separate spatial operations into filter step operations and refinement step operations at the algebraic operator level.

4.2. Combined Refinement

During the mixed query optimization, the geometric computation operation \(G\) of the spatial refinement can be combined with other non-spatial operations to be processed as a unit. According to Observation 1, \(G\) corresponds to the SELECT operation of the relational algebra. Therefore, the \(G\) operation can be combined with other non-spatial SELECT or JOIN operations by Rule 8 or Rule 9.

Example 3 \(\bowtie\) and \(G\) in Equation (2) and Equation (3) can be combined and converted into a \(\bowtie\) operation. Following equation explains that:

\[
\left( \nu \left( \sigma_{a.shape \ s\_intersect \ s\_polygon(x_1, y_1, ..., x_n, y_n)} (\text{buildings : a}) \right) \right)
\bowtie_{a.year=b.year \wedge a.shape \ s\_covered\_by \ s\_polygon(x_1, y_1, ..., x_n, y_n)} (\text{roads : b}) \tag{4}
\]
Since $G$ corresponds to the SELECT operation of the relational algebra, it can also be combined with other $G$ operations in addition to non-spatial operations (see Rule 10). We call the combining of non-spatial operations and/or the $G$ operations of spatial refinement operations the combined refinement. The second strategy for the mixed query optimization is the combined refinement.

**Strategy 2 (Combined Refinement)** Combine the $G$ operations of the spatial refinement operations with non-spatial operations or other $G$ operations by Rule 8 to Rule 10.

### 4.3. Combined Filtering

As in the case of the combined refinement, the spatial filter operation can be combined with other non-spatial filter operations. This can be done by extending the Oid-intersection technique [21] and the Oid-join technique [7, 12, 35] to spatial and non-spatial mixed query processing. Figure 2 and Figure 3 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. The SOA expression for each figure is shown below the figure.

**Example 4** Equation (1) in Example 2 can also be transformed to the following expressions by the Oid-intersection technique:
If the \( R^* \)-tree exists on the attribute “a.shape” and the \( B^+ \)-trees on the attributes “a.year” and “b.year” of both relations, the filter step operations in Equation (5) can be done by the indexing operations. This can be more efficient than the equations in Example 2 and Example 3. \( \square \)

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/or 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\(^1\), 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. When there are many indexes referenced by a query, the combined filtering makes use of 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.

### 4.4. Optimization Rules for Combined Filtering

In the previous sections, we showed the optimization rules for Strategy 1 and Strategy 2. In this section, we will present optimization rules for Strategy 3. Before going to optimization rules for Strategy 3, we will prove the following lemma.

**Lemma 1** If \( F_1 \) and \( F_2 \) are oid-tuple collections which have common attributes and the object duplication in the base class is not allowed,

\[
\nu(F_1) \Join \nu(F_2) = \nu(F_1 \Join F_2)
\]

**Proof.** First, we will prove that \( \nu(F_1) \Join \nu(F_2) \supseteq \nu(F_1 \Join F_2) \). Let \( \rho_i \) and \( \rho_v \) be the join attributes in \( F_1 \Join F_2 \) and \( \nu(F_1) \Join \nu(F_2) \), respectively. As we mentioned in Section 3, the join attributes between object-tuple collections consist of only values. Let \( x \) be an object-tuple in \( \nu(F_1) \) and \( y \) an object-tuple in \( \nu(F_2) \), and let \( m = \iota(x) \) and \( n = \iota(y) \). Let \( m_{\rho_i} \) and \( n_{\rho_i} \) be projections of \( m \) and \( n \) to \( \rho_i \), and let \( x_{\rho_v} \) and \( y_{\rho_v} \) be projections of \( x \) and \( y \) to \( \rho_v \). The necessary and sufficient condition that the pair \( x \) and \( y \) are in \( \nu(F_1 \Join F_2) \) is \( m_{\rho_i} = n_{\rho_i} \). And, if \( m_{\rho_i} = n_{\rho_i} \), then \( \nu(m_{\rho_i}) = \nu(n_{\rho_i}) \), and then \( x_{\rho_v} = y_{\rho_v} \). Therefore the pair \( x \) and \( y \) are also in \( \nu(F_1) \Join \nu(F_2) \).

Next, what remains is to prove that \( \nu(F_1) \Join \nu(F_2) \subseteq \nu(F_1 \Join F_2) \). The necessary and sufficient condition that the pair \( x \) and \( y \) are in \( \nu(F_1 \Join F_2) \) is \( x_{\rho_v} = y_{\rho_v} \). Since object duplication in the base class is not allowed, \( x_{\rho_v} = y_{\rho_v} \) implies \( m_{\rho_v} = n_{\rho_v} \). Therefore, the pair \( x \) and \( y \) are also in \( \nu(F_1) \Join \nu(F_2) \). \( \square \)

In the above lemma, the object duplication means all attribute values are the same between two objects. This means that the same object is inserted twice. Such a case never occurs because we only consider a set among the ODMG collections. As an example for Lemma 1, let \( R \) and \( S \) be

---

\(^1\)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.
the classes of Example 1. Then, both \( \nu(\sigma_{i < 5}^I(R)) \equiv \nu(R \mathbin{\bowtie}_{C=D} S) \) and \( \nu(\sigma_{i < 5}^I(R) \bowtie (R \mathbin{\bowtie}_{C=D} S)) \) are \( \{(i1, (1, 2, 3)), (i6, (3, 1))\} \).

OQL 1 has only two predicates. The combined filtering between two predicates may be simple because such a combining is only an Oid-intersection or Oid-join between two oid-tuple collections resulting from each filter step operation. However, if the number of predicates in a query is more than two, the combined filtering between them may be complicated. A complex query with multiple predicates can be converted to an SOA expression each of whose algebraic operators has only one resulting from each filter step operation. However, if the number of predicates in a query is more because such a combining is only an Oid-intersection or Oid-join between two oid-tuple collections.

Then, both \( \theta_{i < p}^I(R) \) and \( \theta_{i < p}^I(R) \) denote a non-spatial predicate and a spatial predicate for the class \( R \), the class \( S \) and both, respectively. \( \theta_{nsp}(R) \) and \( \theta_{sp}(R) \) denote a non-spatial predicate and a spatial predicate for the class \( R \), and \( \theta_{nsp}(R, S) \) and \( \theta_{sp}(R, S) \) denote a non-spatial predicate and a spatial predicate between the classes \( R \) and \( S \).

**Rule 11** \( \sigma_{\theta_{nsp}}(R) (\nu(F_R)) = \nu(F_R \bowtie (\sigma_{\theta_{nsp}}(R)(R))) \)

**Rule 12** \( \sigma_{\theta_{sp}}(R) (\nu(F_R)) = \theta_{\theta_{sp}}(R) \left( \nu(F_R \bowtie (\sigma_{\theta_{sp}}(R)(R))) \right) \)

**Rule 13** \( \sigma_{\theta_{nsp}}(R, S) (\nu(F_R, S)) = \nu(F_R, S \bowtie (\sigma_{\theta_{nsp}}(R, S)(S))) \)

**Rule 14** \( \sigma_{\theta_{sp}}(R, S) (\nu(F_R, S)) = \theta_{\theta_{sp}}(R, S) \left( \nu(F_R, S \bowtie (\sigma_{\theta_{sp}}(R, S)(S))) \right) \)

**Rule 15** \( \nu(F_R) \bowtie \theta_{nsp}(R, S) \mathbin{\bowtie} E_S = \nu(F_R \bowtie (R \bowtie_{\theta_{nsp}(R, S)} E_S)) \)

**Rule 16** \( \nu(F_R) \bowtie \theta_{sp}(R, S) \mathbin{\bowtie} E_S = \theta_{\theta_{sp}}(R, S) \left( \nu(F_R \bowtie (R \bowtie_{\theta_{sp}(R, S)} E_S)) \right) \)

**Rule 17** \( E_R \bowtie \theta_{nsp}(R, S) \mathbin{\bowtie} \nu(F_S) = \nu((E_R \bowtie_{\theta_{nsp}(R, S)} S) \bowtie F_S) \)

**Rule 18** \( E_R \bowtie \theta_{sp}(R, S) \mathbin{\bowtie} \nu(F_S) = \theta_{\theta_{sp}}(R, S) \left( \nu((E_R \bowtie_{\theta_{sp}(R, S)} S) \bowtie F_S) \right) \)

**Rule 19** \( \nu(F_R) \bowtie \theta_{nsp}(R, S) \mathbin{\bowtie} \nu(F_S) = \theta_{\theta_{sp}}(R, S) \left( \nu(F_R \bowtie (R \bowtie_{\theta_{nsp}(R, S)} S) \bowtie F_S) \right) \)

**Rule 20** \( \nu(F_R) \bowtie \theta_{sp}(R, S) \mathbin{\bowtie} \nu(F_S) = \theta_{\theta_{sp}}(R, S) \left( \nu(F_R \bowtie (R \bowtie_{\theta_{sp}(R, S)} S) \bowtie F_S) \right) \)

**Theorem 1** *The above rules are correct.*

**Proof.** We will only prove Rule 20 which is most complicated. Other rules can be proved in similar manners. Since \( F_R \) is an oid-tuple collection resulting from the previous filter step operators, the
number of the distinct objects for the class \( R \) in \( \nu(F_R) \) is less than or equal to that of the base class. The same condition holds between the class \( S \) and \( \nu(F_S) \). Therefore,

\[
\nu(F_R) \precsim_{\theta_{i,p}(R,S)} \nu(F_S) = \left( \nu(F_R) \precsim R \right) \precsim_{\theta_{i,p}(R,S)} \left( S \precsim \nu(F_S) \right)
\]

\[
= \nu(F_R) \precsim \left( (R \precsim_{\theta_{i,p}(R,S)} S) \precsim \nu(F_S) \right) \quad \text{by join associativity}
\]

\[
= \nu(F_R) \precsim \left( (\sigma_{\theta_{i,p}(R,S)}(\nu(R \precsim_{\theta_{i,p}(R,S)} S))) \precsim \nu(F_S) \right) \quad \text{by Rule 4}
\]

\[
= \sigma_{\theta_{i,p}(R,S)}(\nu(F_R) \precsim \left( (R \precsim_{\theta_{i,p}(R,S)} S) \precsim \nu(F_S) \right)) \quad \text{by select join commutativity (Rule 6)}
\]

\[
= \sigma_{\theta_{i,p}(R,S)} \left( \nu(F_R) \precsim \left( (R \precsim_{\theta_{i,p}(R,S)} S) \precsim \nu(F_S) \right) \right) \quad \text{by Lemma 1}
\]

In Example 4, we already showed the application of Rule 15. That is, Equation (5) is derived by applying Rule 15 to Equation (3). We will give another example to show the application of other rules to a complex query.

**Example 5** Consider the following OQL query:

**OQL 2**

```
select * from a in buildings, b in roads, c in districts where
a.shape s_intersect b.route and a.shape s_covered by c.boundary and
a.year < 1980 and c.boundary s_intersect s_polygon(x_1, y_1, ..., x_n, y_n);
```

Let \( \theta_1, \theta_2, \theta_3 \) and \( \theta_4 \) be \( a.shape s_intersect b.route, a.shape s_covered by c.boundary, a.year < 1980 \) and \( c.boundary s_intersect s_polygon(x_1, y_1, ..., x_n, y_n) \), respectively. Equation (6) is an SOA-expression for OQL 2.

\[
\left( (\sigma_{\theta_{a,b}}(\text{buildings})) \precsim_{\theta_{b,c}} \text{roads} \right) \precsim_{\theta_{a,d}} (\sigma_{\theta_{c,d}}(\text{districts}))
\]

(6)

The following equations are some ISOA expressions generated by the ESFAR optimization rules. By applying Rule 2 and Rule 1 to \( \sigma_{\theta_{a,b}} \) and \( \sigma_{\theta_{c,d}} \) of Equation (6), respectively,

\[
\left( (\nu(\sigma_{\theta_{a,b}}(\text{buildings}))) \precsim_{\theta_{b,c}} \text{roads} \right) \precsim_{\theta_{a,d}} (\sigma_{\theta_{c,d}}(\nu(\sigma_{\theta_{b,c}}(\text{districts}))))
\]

(7)

By applying Rule 16 to \( \precsim_{\theta_{b,c}} \),

\[
G_{\theta_{b,c}} \left( \left( (\nu(\sigma_{\theta_{a,b}}(\text{buildings}))) \precsim (\text{roads} \precsim_{\theta_{b,c}} \text{roads}) \right) \right) \precsim_{\theta_{a,d}} (\sigma_{\theta_{c,d}}(\nu(\sigma_{\theta_{b,c}}(\text{districts}))))
\]

(8)

By applying Rule 6 and Rule 10 to \( \precsim_{\theta_{b,c}} \), \( G_{\theta_{b,c}} \), and \( \sigma_{\theta_{c,d}} \),

\[
G_{\theta_{b,c} \precsim_{\theta_{b,c} \precsim_{\theta_{b,c}}}} \left( \nu \left( (\sigma_{\theta_{a,b}}(\text{buildings})) \precsim (\text{roads} \precsim_{\theta_{b,c}} \text{roads}) \right) \right) \precsim_{\theta_{a,d}} (\sigma_{\theta_{c,d}}(\nu(\sigma_{\theta_{b,c}}(\text{districts}))))
\]

(9)

By applying Rule 20 and Rule 10 to \( \precsim_{\theta_{b,c}} \) and \( G_{\theta_{b,c} \precsim_{\theta_{b,c} \precsim_{\theta_{b,c}}}} \),

\[
G_{\theta_{b,c} \precsim_{\theta_{b,c} \precsim_{\theta_{b,c}}}} \left( \nu \left( (\sigma_{\theta_{a,b}}(\text{buildings})) \precsim (\text{roads} \precsim_{\theta_{b,c}} \text{roads}) \right) \right)
\]

\[
\precsim \left( (\text{roads} \precsim_{\theta_{b,c}} \text{roads}) \precsim (\sigma_{\theta_{c,d}}(\text{districts}))) \right)
\]

(10)
4.5. Further Optimization on Combined Filtering

Combining the spatial filter steps can be done by other methods than the Oid-join. If the same attributes exist on the spatial predicates of an input query and these predicates can be evaluated by the combined filtering, the filter step operations can be performed in a single algorithm. In Example 5, “c.boundary” is the common spatial attribute which appears in both a spatial select predicate ($\theta_1$) and a spatial join predicate ($\theta_2$). If a spatial select and a spatial join have a common spatial attribute and the $R^k$-tree indexes exist on all the attributes these operations refer to, the SSF ($R$-tree-select) and SJF ($R$-tree-join) operations can be combined into a single algorithm as shown in Figure 4. We call this combined filtering the $R$-tree-select-join.

The NSF ($B$-tree-select) and NJF ($B$-tree-join) operations can also be combined in a manner similar to the $R$-tree-select-join. We call the combined filtering between $B$-tree-select and $B$-tree-join the $B$-tree-select-join.

**Strategy 4 ($R$-tree-select-join)** Combine the $R$-tree-select and $R$-tree-join operations at the combined filter step into a $R$-tree-select-join if a common attribute appears in both a spatial select predicate and a spatial join predicate.

The common spatial attributes can also appear between the spatial join predicates. In Example 5, two spatial join predicates $\theta_1$ and $\theta_2$ have a common spatial attribute “a.shape”. If two spatial join operations have a common spatial attribute and the $R^k$-tree indexes exist on all the attributes these operations refer to, the two $R$-tree-joins can be combined into a 3-way $R$-tree-join as shown in Figure 5.
When there are several join predicates and they can be connected into a graph by common attributes, we call the connection of join operations by common attributes the common attribute join graph (CAJG). If there is a spatial CAJG and the R*-tree indexes exist on all the join attributes in the CAJG, the R-tree-join operations can be combined into the M-way R-tree-join. Recently, several M-way R-tree-join algorithms were proposed [20, 24, 26]. The M-way B-tree-join can be performed in a manner similar to the M-way R-tree-join.

**Strategy 5 (M-way R-tree-join)** Combine the R-tree-join operations into an M-way R-tree-join at the combined filter step if there is a spatial CAJG between the spatial join operations.

The R-tree-select-join and M-way R-tree-join operations can also be combined into the M-way R-tree-select-join if they have a common attribute. Likewise, the M-way B-tree-select-join can also be done in a similar manner.

**Example 6** If we apply Strategy 4 and Strategy 5 to Equation (10) of Example 5, the following expression can be generated:

$$G_{\theta_1 \land \theta_3 \land \theta_2} \left( \nu \left( \text{B-tree-select}_{\theta_3}(\text{buildings}) \right) \right) \land \left( \text{M-way R-tree-select-join}_{\theta_3 \land \theta_2 \land \theta_4} \left( \text{buildings}, \text{roads}, \text{districts} \right) \right)$$

We have suggested five optimization strategies and their related optimization rules for spatial and non-spatial mixed queries in this section. However, these strategies by themselves cannot always generate the most efficient plan. Therefore, we should check each strategy based on a cost model for the input query. The implementation and experiments of these optimization strategies and rules are given in [28, 29] and the next section of this paper.

5. IMPLEMENTATION AND EXPERIMENTS

To measure the performance of the ESFAR optimization strategies, we implemented an optimizer using the Volcano optimizer generator (VOG) [14] and execution algorithms of the ISOA operators described in Section 3 and Section 4. Since the simulation results for the optimizer using a cost model were presented in the early version of this paper [28, 29], we concentrate on real execution for the ESFAR strategies. The experiments were performed on SUN Ultra II 170 MHz workstation with 384 MB main memory which Solaris 2.5.1 was running on. The page size and the number of LRU buffers were fixed to 4 KB and 256, respectively. The experimental data is the TIGER/Line data [34] extracted from roads and hydrographies of three counties of the California state. We built the B-tree on the TLID (TIGER/Line Identification number) field [34] as the primary index and the R*-tree on the line segment field as a secondary index for each TIGER data. The statistics of the TIGER data used in these experiments are summarized in Table 2 (see Appendix A for all subsequent tables). A and H, respectively, represent CFCC (Census Feature Class Code) [34] for roads and hydrographies in the TIGER data in Table 2.

The query types used for these experiments are “non-spatial select and spatial select”, “spatial select and non-spatial join” and “3-way spatial join”. We compare the actual response time of execution plans which are generated by the traditional (TRA) method and the ESFAR method. Actually, the ESFAR optimizer always generates more efficient execution plans than the traditional optimizer. This is because the operators and rules which the ESFAR optimizer uses include those of the traditional optimizer. If the execution plan generated by the rules of Section 4 is more expensive than that only by the traditional rules, the ESFAR optimizer chooses the execution plan by the traditional rules. To measure the net effects of the ESFAR strategies, however, we compare the execution plans only by the traditional method and those only by the ESFAR method.

The first experiment was performed for the following “non-spatial select and spatial join” type of query:
OQL 3 select a from a in roads where a.LINE s.intersect s.polygon(x₁,y₁,…,xₙ,yₙ) and a.TLID > z;

In the above query, s.polygon(x₁,y₁,…,xₙ,yₙ) is a constant polygon, and z an integer constant. The number of points of the constant polygon is 13.

Since the B+ tree exists on the non-spatial attribute and the R*-tree on the spatial attribute, the traditional optimizer will generate one of the following execution plans: TRA1 is a plan that the non-spatial select is first applied using the B+ tree, then both the filter and refinement steps of the spatial select for the result from the non-spatial select. TRA2 is a plan that both the filter and refinement steps of the spatial select are first applied using the R*-tree, then the non-spatial select for the result from the spatial select. Plans using no index such as the Seq-select will never be selected by the query optimizer because the spatial or non-spatial select using an index is much cheaper unless the selectivity is extremely large (e.g., more than 50%).

- TRA1: B-tree-select – spatial select
- TRA2: R-tree-select with refinement step – non-spatial select

The ESFAR optimizer will generate one of the following execution plans, in addition to the above plans, which are generated by the ESFAR strategies:

- ESFAR1: R-tree-select – non-spatial select – spatial select refinement (Strategy 1)
- ESFAR2: R-tree-select – combined refinement (Strategy 2)
- ESFAR3: (B-tree-select, R-tree-select) – Oid-intersect – spatial select refinement (Strategy 3)

Table 3 shows the query response time for the traditional method and the ESFAR method with the varying non-spatial selectivity (S_{nsp}) and spatial selectivity (S_{sp}) (Boxed items represent the fastest response time). The selectivity of the non-spatial select is controlled by the integer constant z and the selectivity of the spatial select by the size of the constant polygon s.polygon(x₁,y₁,…,xₙ,yₙ). In the sequel, the selectivity of the spatial select considers only the filter step by MBR. The average hit ratio by the refinement step was observed to be approximately 60%. The experimental result shows, when S_{nsp} is high and S_{sp} is low or both are similar, ESFAR has benefits. Especially, if S_{nsp} – S_{sp} is large, ESFAR2 performs best. On the other hand, if the difference is small, ESFAR3 using the Oid-intersection technique has the best performance.

If S_{nsp} is higher than S_{sp}, executing the non-spatial operation first (TRA1) shows the best performance. Like this, a plan from the traditional method is sometimes more efficient than that from the ESFAR method. However, it does not matter because as we mentioned, if the estimation cost of the plan from the ESFAR method is more expensive than that from the traditional method, the ESFAR optimizer will select the traditional plan.

Next, we conducted an experiment for the following query type which consists of a spatial select and a non-spatial join:

OQL 4 select a,h from a in roads, h in hydrographies where a.LINE s.intersect s.polygon(x₁,y₁,…,xₙ,yₙ) and -z ≤ a.TLID-b.TLID ≤ z;

We measured the response time of several execution plans from the above query with the varying spatial selectivity and non-spatial selectivity. The non-spatial join selectivity is determined by the range value z in the above query. The execution plans from the traditional optimizer are the following:

- TRA1: R-tree-select with refinement step – indexed nested loop join
- TRA2: B-tree-join – spatial select using oid-pairs (S_{ObjPair.Select})

The execution plans from the ESFAR optimizer are the following in addition to the above plans:

- ESFAR1: R-tree-select – indexed nested loop join – spatial select refinement (Strategy 1)
**ESFAR2**: R-tree-select – combined refinement of indexed nested loop join and spatial select (Strategy 2)

**ESFAR3**: (R-tree-select, B-tree-join) – Oid-join – spatial select refinement using oid-pairs (Strategy 3)

In the above plans, we did not include the non-spatial join algorithms using no index such as the hybrid hash join. If the join selectivity is low, the B-tree-join is known to be more efficient than the hybrid hash join [35]. Although the join selectivity is high, since the spatial select considerably reduces the intermediate result size for the outer class, and one index still remains in the inner class, the indexed nested loop join is more efficient, in this case, than the hybrid hash join [33]. Therefore, the optimizer will seldom select the plan having the non-spatial join algorithms using no index as an optimal plan.

Table 5 shows the result for the above query. Contrary to the case of “non-spatial select and spatial select”, the ESFAR strategies have many effects in the case of the high spatial selectivity and the low non-spatial selectivity. This is because if the non-spatial join selectivity is low, many objects which are supposed to be refined by the spatial select are filtered out by the join. In this case, if \( z \) is low, ESFAR3 performs best. Otherwise, ESFAR2 has the best performance. This is because if \( z \) is high, the number of oid-pairs is large and consequently the Oid-join cost is high. The numbers of oid-pairs resulting from the non-spatial join for cases \( z = 1, 5 \) and 25 are 11740, 53763 and 290827, respectively. In the case of the low spatial selectivity \( (S_{sp} = 0.01) \), TRA1 has the fastest response time.

Table 6 shows the response time for the above query when the spatial select is applied to the hydrography data which is the smaller class and whose spatial attribute is more complex than the road data. The result is similar to Table 5 except the following: If \( z \) is 25, executing spatial selection first (TRA1) always performs best because the result size from the spatial select is much smaller than the join size. If \( z \) is 1, even though the spatial selectivity is low \( (S_{sp} = 0.01) \), ESFAR1 and ESFAR2 perform better than TRA1. This is because since the hydrography data is more complex than the road, the refinement cost per object is more expensive than the case of the road, therefore the filtering effect by the non-spatial join is higher.

Finally, we measured the response time for the following 3-way spatial join:

**OQL 5**: select * from a in \( R \), b in \( S \), c in \( T \) where a.LINE \( s \_intersect \) b.LINE and b.LINE \( s \_intersect \) c.LINE;

The 3-way spatial join was performed for the road data and the hydrography data of 3 counties, respectively. To join different county regions, we subtracted the coordinate of the center point of each county from the x and y coordinates of the original TIGER data. An execution plan from the traditional optimizer is:

- **TRA**: R-tree-join between the first two classes – indexed nested loop join

The execution plans from the ESFAR optimizer are the following in addition to the above plan:

- **ESFAR3**: (R-tree-join, R-tree-join) – Oid-join – combined refinement using oid-tuples \( (S_{ObjTupleSelect}) \) (Strategy 2 and Strategy 3)
- **ESFAR5**: 3-way R-tree-join – combined refinement using oid-tuples (Strategy 2 and Strategy 5)

We used the algorithm in [26] as an M-way R-tree join for ESFAR5. Since the above query includes only spatial predicates, Strategy 1 by itself does not contribute to reduce the response time. Therefore, we excluded ESFAR1 in this experiment. The response time for the above query is shown in Table 7. In most cases, ESFAR has the faster response time than TRA. This is because the number of refinement operations in ESFAR is smaller than that in TRA. Therefore, we expect that if the spatial objects are more complex, the effect of the ESFAR strategy becomes bigger. The numbers of refinement operations in TRA and ESFAR are shown in Table 8. The average hit ratio by the spatial join refinement was observed to be approximately 25%. In a join among
roads, i.e., Riv:San.Ker(A), ESFAR performs worse than TRA. This depends on the number of oid-tuples resulting from the combined filtering (see Table 9). If the intermediate result size from the combined filtering is large, there is a large overhead for reading oid-tuples and the relevant objects. In comparison of ESFAR3 and ESFAR5, if the number of oid-tuples from the combined filtering is small, ESFAR5 performs better. Otherwise, ESFAR3 performs better. This is because ESFAR3 just uses the sorted oid-tuples resulting from the Oid-join\(^1\) while ESFAR5 sorts the oid-tuples from the combined filtering to avoid redundant refinement operations before the combined refinement.

In plan TRA, we used the R-tree-join as the first join algorithm because the R-tree-join is known to be most efficient when both inputs have the R*-trees [30]. And we used the indexed nested loop join as the second join. Of course, other spatial join algorithms such as the Seeded-Tree join [18] or the Spatial-Hash join [19] can be used instead of the indexed nested loop join. However, main performance differences among them are in the filter step. On the other hand, a large portion of the response time of the spatial join is consumed in the refinement step. Therefore, we believe that although the second join algorithm is changed to another, the total response time will not be changed so much.

6. CONCLUSIONS

In this paper, we proposed new optimization strategies which exploit the two step processing of a spatial query. The main idea is to start the two-step processing of the spatial query, which has been applied only in the query execution phase, from the query optimization phase. We showed that filter and refinement steps 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 and the execution algorithms. Through experiments using real data, we showed that our ESFAR optimization technique generated more efficient execution plans than those by traditional optimization techniques in many cases. It was also observed that the traditional method performed better than the ESFAR method in some cases. However, it does not matter because if the estimation cost for the ESFAR method is higher than that for the traditional method, the query optimizer will select an execution plan by the traditional method.

We used the exhaustive search strategy by the dynamic programming built in VOG as an optimization method. As a future study, we will develop a heuristic algorithm for the ESFAR optimization technique.

Acknowledgments — We would like to thank Yannis Ioannidis, area editor, for his help and anonymous referees for their valuable comments. This research was supported by the Software Technology Enhancement Program 2000 of the Ministry of Science and Technology of Korea and the Basic Research Program (grant number 99-2-315-001-3) of the Korea Science and Engineering Foundation.

REFERENCES


\(^1\)We simply use the sort-merge join strategy as the Oid-join.


APPENDIX A

<table>
<thead>
<tr>
<th>county</th>
<th># of obj</th>
<th>avg. # of pts</th>
<th>MBR density</th>
<th>domain area</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>H</td>
<td>A</td>
<td>H</td>
<td>(10^5)</td>
</tr>
<tr>
<td>Kern</td>
<td>113407</td>
<td>26014</td>
<td>3.7</td>
<td>8.9</td>
<td>0.26</td>
</tr>
<tr>
<td>Riverside</td>
<td>91751</td>
<td>15440</td>
<td>3.5</td>
<td>12.2</td>
<td>0.21</td>
</tr>
<tr>
<td>San Bernadino</td>
<td>146026</td>
<td>28187</td>
<td>3.5</td>
<td>12.4</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Table 2: The Statistics of the TIGER Data

<table>
<thead>
<tr>
<th>Snep</th>
<th>Step</th>
<th>TRA1</th>
<th>TRA2</th>
<th>ESFAR1</th>
<th>ESFAR2</th>
<th>ESFAR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.20</td>
<td>0.05</td>
<td>1648</td>
<td>5819</td>
<td>2488</td>
<td>2276</td>
<td>1523</td>
</tr>
<tr>
<td>0.01</td>
<td>0.00</td>
<td>1173</td>
<td>296</td>
<td>153</td>
<td>142</td>
<td>438</td>
</tr>
<tr>
<td>0.05</td>
<td>0.05</td>
<td>543</td>
<td>5813</td>
<td>2070</td>
<td>1990</td>
<td>850</td>
</tr>
<tr>
<td>0.01</td>
<td>0.00</td>
<td>341</td>
<td>296</td>
<td>144</td>
<td>136</td>
<td>155</td>
</tr>
<tr>
<td>0.20</td>
<td>0.05</td>
<td>96</td>
<td>5806</td>
<td>1806</td>
<td>1793</td>
<td>527</td>
</tr>
<tr>
<td>0.01</td>
<td>0.00</td>
<td>65</td>
<td>291</td>
<td>121</td>
<td>119</td>
<td>67</td>
</tr>
</tbody>
</table>

Table 3: Response Time for “Non-Spatial Select and Spatial Select” Type of Query for Road Data of Riverside County
Table 4: Response Time for “Non-Spatial Select and Spatial Select” Type of Query for Hydrography Data of Riverside County

<table>
<thead>
<tr>
<th>S_{sp}</th>
<th>z</th>
<th>TRA1</th>
<th>TRA2</th>
<th>ESFAR1</th>
<th>ESFAR2</th>
<th>ESFAR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.20</td>
<td>1</td>
<td>9311</td>
<td>3620</td>
<td>7737</td>
<td>7164</td>
<td>2345</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>10084</td>
<td>5872</td>
<td>10768</td>
<td>8384</td>
<td>4449</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>14197</td>
<td>15774</td>
<td>27206</td>
<td>13414</td>
<td>15294</td>
</tr>
<tr>
<td>0.05</td>
<td>1</td>
<td>2344</td>
<td>3348</td>
<td>2108</td>
<td>1921</td>
<td>1384</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2529</td>
<td>4870</td>
<td>2979</td>
<td>2211</td>
<td>2599</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>3448</td>
<td>11488</td>
<td>7533</td>
<td>3372</td>
<td>9267</td>
</tr>
<tr>
<td>0.01</td>
<td>1</td>
<td>425</td>
<td>3265</td>
<td>505</td>
<td>449</td>
<td>1100</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>494</td>
<td>4644</td>
<td>789</td>
<td>558</td>
<td>2081</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>835</td>
<td>10651</td>
<td>2180</td>
<td>941</td>
<td>7903</td>
</tr>
</tbody>
</table>

Table 5: Response Time for the Query of “Spatial Select and Non-Spatial Join” Type for Riverside County when the Spatial Select is Performed on Road Data

<table>
<thead>
<tr>
<th>S_{sp}</th>
<th>z</th>
<th>TRA1</th>
<th>TRA2</th>
<th>ESFAR1</th>
<th>ESFAR2</th>
<th>ESFAR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.20</td>
<td>1</td>
<td>3296</td>
<td>3789</td>
<td>2989</td>
<td>2938</td>
<td>9087</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3917</td>
<td>6015</td>
<td>5745</td>
<td>3778</td>
<td>4231</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>7367</td>
<td>14444</td>
<td>19384</td>
<td>7622</td>
<td>13773</td>
</tr>
<tr>
<td>0.05</td>
<td>1</td>
<td>816</td>
<td>3373</td>
<td>788</td>
<td>661</td>
<td>1284</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>961</td>
<td>4845</td>
<td>1527</td>
<td>978</td>
<td>2054</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>1650</td>
<td>10330</td>
<td>4900</td>
<td>1796</td>
<td>8806</td>
</tr>
<tr>
<td>0.01</td>
<td>1</td>
<td>170</td>
<td>3258</td>
<td>150</td>
<td>137</td>
<td>1005</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>200</td>
<td>4511</td>
<td>279</td>
<td>199</td>
<td>1965</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>330</td>
<td>9431</td>
<td>849</td>
<td>352</td>
<td>7556</td>
</tr>
</tbody>
</table>

Table 6: Response Time for the Query of “Spatial Select and Non-Spatial Join” Type for Riverside County when the Spatial Select is Performed on Hydrography Data

<table>
<thead>
<tr>
<th>R:S:T(A)</th>
<th>TRA</th>
<th>ESFAR3</th>
<th>ESFAR5</th>
<th>R:S:T(H)</th>
<th>TRA</th>
<th>ESFAR3</th>
<th>ESFAR5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ker:Riv:San</td>
<td>33037</td>
<td>25794</td>
<td>26245</td>
<td>Ker:Riv:San</td>
<td>73235</td>
<td>65777</td>
<td>67808</td>
</tr>
<tr>
<td>Riv:Ker:San</td>
<td>27418</td>
<td>17328</td>
<td>15663</td>
<td>Riv:Ker:San</td>
<td>39624</td>
<td>27567</td>
<td>26787</td>
</tr>
<tr>
<td>Riv:San:Ker</td>
<td>38801</td>
<td>48196</td>
<td>147603</td>
<td>Riv:San:Ker</td>
<td>54236</td>
<td>48248</td>
<td>51821</td>
</tr>
</tbody>
</table>

Table 7: Query Response Time for the Query of the “3-Way Spatial Join” Type
Table 8: The Number of Refinement Operations for “3-Way Spatial Join”

<table>
<thead>
<tr>
<th>R:S:T(A)</th>
<th>TRA</th>
<th>ESFAR</th>
<th>R:S:T(H)</th>
<th>TRA</th>
<th>ESFAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ker:Riv:San</td>
<td>60229</td>
<td>33871</td>
<td>Ker:Riv:San</td>
<td>39294</td>
<td>32108</td>
</tr>
<tr>
<td>Riv:Ker:San</td>
<td>58065</td>
<td>26200</td>
<td>Riv:Ker:San</td>
<td>31038</td>
<td>19104</td>
</tr>
<tr>
<td>Riv:San:Ker</td>
<td>127777</td>
<td>92327</td>
<td>Riv:San:Ker</td>
<td>43492</td>
<td>35554</td>
</tr>
</tbody>
</table>

Table 9: The Number of Oid-Tuples from the Combined Filtering for “3-Way Spatial Join”

<table>
<thead>
<tr>
<th>R:S:T(A)</th>
<th># of oid tuples</th>
<th>R:S:T(H)</th>
<th># of oid tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:S</td>
<td>S:T</td>
<td>R:S:T</td>
<td>R:S</td>
</tr>
<tr>
<td>Ker:Riv:San</td>
<td>46722</td>
<td>53096</td>
<td>111855</td>
</tr>
<tr>
<td>Riv:Ker:San</td>
<td>46722</td>
<td>49494</td>
<td>68833</td>
</tr>
<tr>
<td>Riv:San:Ker</td>
<td>53096</td>
<td>49494</td>
<td>1888979</td>
</tr>
</tbody>
</table>