An Effective Method for Processing Queries with Expensive Predicates

Hanxiong Chen
Institute of Industrial Information,
Tsukuba International University
Manabe 6, Tsuchiura-shi, Ibaraki 300-0051, Japan

1 Introduction

In this paper, we formalize a query processing scheme based on the filtering for the selection operation involving a computationally expensive predicate. A cost model is provided and an algorithm is given to determine an optimal query evaluation plan. Experiments on querying GIS illustrate the effect. Some empirical results are shown.

1.1 Motivation

Recently, there has been increasing interest in geography and geographic information system (GIS). A geographic database is a combination of spatial and non-spatial data, with complex data structures and analyses. Due to the special properties of geographical information, query patterns used in GIS are different from other non-spatial applications. As pointed in [Worb95], traditional DBMS's are unsuitable for spatial data management. One of the reasons is the indexes required for spatial data are not supported by proprietary DBMS straighforward.

Four spatial objects o_1, o_2, o_3, o_4 and their Minimum Bounding Rectangle (MBR) are shown in Figure 1. The query is to find objects which ovelap o_1. Instead of execute the overlaping operation between o_2, o_3, o_4 and o_1 immediately, filters are applied. For example, MBR is used to exclude o_3, because its MBR does not overlap that of o_1, so it is impossible that o_3 itself overlaps o_1. Other filters can also be used to exclude o_2. Accordingly, we only need to investigate o_4.

To jugde the overlaping relationship between two MBR's, it needs only to compare two pairs of coordinates, hence the cost is negligible.

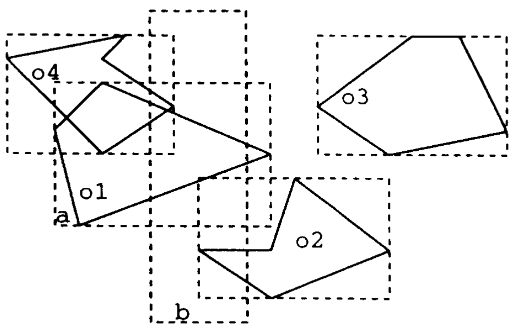

Figure 1: example of overlap

Denoting the overlap relationship between two given spatial objects x, y by $overlap(x, y)$, and the overlap relationship between the MBRs of x, y by $MBRoverlap(x, y)$, we have

$$overlap(x, y) \leftrightarrow MBRovlap(x, y) \wedge overlap(x, y) \quad (1)$$

Further more, let the MBR of o_1 be a, supposing that there is a rectangle b which crosses over a as shown in the figure above. If there is any object x which has b as its MBR, Then this object x must overlaps o_1. For such x, there is no need to evaluate $overlap(x,\ o_1)$.

Let the cross relationship between the MBRs of x, y be $MBRcross(x, y)$, then (1) becomes

$$\begin{aligned} overlap(x, y) \leftrightarrow MBRcross(x, y)\ \vee \\ (MBRovlap(x, y) \wedge overlap(x, y)) \end{aligned} \quad (2)$$

Now it can be seen that using predicates $MBRcross(x, y)$ and $MBRovlap$ as filters, it is possible to figure out whether some objects overlap with other objects without evaluating the predicate $overlap$. Our purpose is to find the optimal application of such filters.

1.2 Related Works

Query processing of expensive predicates has been discussed in [CYY+92], It comes from earlier concept of ADT function in the database context [Atk87, Sto88], and the concept of extensible query processing[Haa89, PHH92].

Query rewriting is known as an efficient method for processing query, and [PHH92, CS93] use rewrite rule. The disadvantage of these works is that their passively rewrite the order of the given predicates. In contrast, our approach takes disjunctive queries into consideration, and it actively adds "cheaper" predicates which do not appear in the query originally.

Orthogonal issues of predicate placement are discussed in [KBZ86, YKY+91, HS93, Hel94, CS96]. Comparing to these works which treat more than one predicate, we focus on filtering for single predicate. This narrow focus enabled us to clarify the relationship between filters.

Using index structure such as R-tree ([Gutt84]) is another orthogonal issue. A typicall query to a R-tree index structure is to find all index records in the R-tree whose rectangles overlap a search rectangle. As shown in the example above, rectangles overlap is nothing more than a filter in our framework. This is why we say that a method using R-tree is orthogonal. Simillarly, other examples index which can be used in GIS in [PTSE95] may be available as filters in our framework.

2 Formalization

In this section, we describe the notations which will be used in the following sections. Let R be a set of objects, and let f_i and f be a predicate. We use $\hat{f}$ to indicate a predicate if its computational cost is very high. Furthermore, let op be either a logical conjunction operator ($\wedge$) or a logical disjunction operator ($\vee$).

The main goal of our research is to minimize the total computational cost for evaluating a simple selection query like $\sigma_{\hat{f}}R$. In order to achieve this, we attempt to use a set of predicates connected with logical operators as follows:

$$\sigma_{f_1\ op\ f_2\ op\ \ldots\ op\ f_n}R \equiv \sigma_{\hat{f}}R$$

Here, we call $(f_1\ op\ f_2\ op\ ...\ op\ f_n)$ an *evaluation plan* of $\hat{f}$ in the sense that both queries give the same results for a given relation R. It is worth noting that each f_i above acts as a "filter" to reduce the number of objects to be evaluated using $\hat{f}$. We use $\overrightarrow{f}_i$, $\overleftarrow{f}_i$ and $\overleftrightarrow{f}_i$ to indicate a predicate f_i if it is a necessary, sufficient, and necessary-sufficient condition of $\hat{f}$, respectively.

Example 1 *For the example given in the previous section, let* o_1 *be a constant and* x *be a tuple variable. Furthemore, let* $\hat{f}(o_1, x) = overlap(o_1, x)$, $\overleftarrow{f}_1(o_1, x) = MBRcross(o_1, x)$, $\overrightarrow{f}_2(o_1, x) = MBRoverlap(o_1, x)$, *and* $\overleftrightarrow{f}_3(o_1, x) = overlap(o_1, x)$. *Then (2) can be rewritten as*

$$\hat{f}(o_1, x) \equiv \overleftarrow{f}_1(o_1, x) \vee \overrightarrow{f}_2(o_1, x) \wedge \overleftrightarrow{f}_3(o_1, x)$$

In term of performance, for a given object $x \in R$, it can be a significant saving if either $\overleftarrow{f}_1$ is true or $\overrightarrow{f}_2$ is false, since we can avoid evaluating $\hat{f}$. Some comments can be briefly made in terms of an evaluation plan of $\hat{f}$.

- The evaluation plain will be evaluated from left to right in order.
- We assume that all $\overrightarrow{f}_i$ and $\overleftarrow{f}_i$ will be evaluated in a polynomial time.
- In any evaluation plan, if there is a predicate $\overleftrightarrow{f}_i$ then $\overleftrightarrow{f}_i = \hat{f}$. There must be at least one of such $\overleftrightarrow{f}_i$ predicates included which will be evaluated last if needed. If there are two $\overleftrightarrow{f}_i$ and $\overleftrightarrow{f}_j$ included, then only one of them needs to be evaluated. This guarantees that $\hat{f}$ will never be executed twice for an object in the relation R.

Two normal forms are considered: namely the disjunctive normal form and the conjunctive normal form.

$$\bigvee_i (\bigwedge_j f_{k_{ij}}) \tag{3}$$

and

$$\bigwedge_i (\bigvee_j f_{k_{ij}}) \tag{4}$$

where each $f_{k_{ij}}$ indicates the predicate f_k to be evaluated as the j-th predicate in the i-th literal. Since the cost relies on the order of the execution of the evaluation plan, the evaluation order in either of the two normal forms is as follows: (a) evaluate all the predicates in the first literal, $i = 1$, in the order $j = 1, 2, \cdots$; and (b) repeat it for the next literal. For simplicity, we use the following notation $\lambda_i = (f_k, f_l, \cdots)$ to denote the i-th literal in the disjunctive normal form. All $(f_k, f_l, \cdots)$ are connected using the logical operator $\wedge$ and will be evaluated from left to right. Also, we use the following notation $\nu_i = (f_k, f_l, \cdots)$ to denote the i-th literal in the conjunctive normal form. In this case, all $(f_k, f_l, \cdots)$ are connected using the logical operator $\vee$ and will be evaluated from left to right as well.

Two issues

The evaluation plan can be either of the two normal forms. Here, we are concerned with local optimization and global optimization for an evaluation plan. By local optimization, we mean two assertions. Firstly, we mean a permutation of all predicates in either λ_i or ν_i. This permutation will be optimal in the sense that computational cost of either λ_i or ν_i will be minimized when it is applied to a relation R. Secondly, we mean some predicates in either λ_i or ν_i will be pruned. By global optimization, we mean the order of evaluation of the literals and further pruning if needed.

3 Local Optimization

In this section, we discuss local optimization. In next subsection, we describe briefly our previous work [CYY+92] for optimizing the literal $\lambda_i = (f_k, f_l, \cdots)$ in a disjunctive normal form. In the following section, we will discuss how to optimize the literal $\nu_i = (f_k, f_l, \cdots)$ in a conjunctive normal form. The two issues mentioned above are taken into consideration: determination of the order of evaluation for all the predicates and pruning unnecessary predicates in either λ_i or ν_i. The parameters used in the following discussions are given below. Let $\hat{c}$ and c_i are the computational costs for the predicate $\hat{f}$ and f_i, respectively. Let I_{f_i} be a selectivity for a predicate f_i as $I_{f_i} = |\sigma_{f_i}R|/|R|$ where $|R|$ denotes the cardinality of R. In a similar fashion, $I_{\hat{f}} = |\sigma_{\hat{f}}R|/|R|$. Here, we assume that the selectivity of f_j is independent of any others.

3.1 Conjunctive Filtering

The literal $\lambda_i = (f_k, f_l, \cdots)$ in a disjunctive normal form must be one of the following two forms:

- If there is only one predicate in λ_i, then $\lambda_i = (\overrightarrow{f_k})$ or $\lambda_i = (\overleftrightarrow{f_k})$.[1]
- If there are more than one predicate in λ_i, then

  $$\lambda_i = (\overrightarrow{f_k}, \overrightarrow{f_l}, \cdots, \overrightarrow{f_m}, \overleftrightarrow{f_n})$$

  First, there is no sufficient condition of $\hat{f}$ in λ_i in this case. Second, all the predicates except for the last to be evaluated are necessary conditions of $\hat{f}$.

In addition, given two literal λ_i and λ_j for $i \neq j$, the two literals must be different in the sense that they don't contain the same set of predicates. It is worth noting that $\sigma_{\lambda_i}R \equiv \sigma_{\hat{f}}R$ holds for any literal λ_i except for the case $\lambda_i = (\overrightarrow{f_k})$. To estimate the cost of λ_i, a reduction ratio, called "false drop", is introduced for a necessary predicate $\overrightarrow{f_k}$, denoted by d_k.

$$d_k = (|\sigma_{f_k}R| - |\sigma_{\hat{f}}R|)/(|R| - |\sigma_{\hat{f}}R|) = (I_{f_k} - I_{\hat{f}})/(1 - I_{\hat{f}})$$

Since $\overrightarrow{f_k}$ is a necessary condition of $\hat{f}$, then $0 < I_{\hat{f}} < I_{f_k} \leq 1$. Hence, $0 < d_k \leq 1$. Figure 2 illustrates these relationships, where "fi" is a simplified notation of $\sigma_{f_i}(R)$. Ideally, a smaller false drop is desirable.

Let's only consider the evaluation of the literal $\lambda_2 = (\overrightarrow{f}_2\,(o_1, x), \overleftrightarrow{f}_3\,(o_1, x))$ in Example 1.[2]

$$\sigma_{\hat{f}(o_1,x)}R \equiv \sigma_{(\overrightarrow{f}_2(o_1,x), \overleftrightarrow{f}_3(o_1,x))}R$$

[1] $\lambda_i = (\overrightarrow{f_k})$ is only allowed if there is λ_j for $j \neq i$, in which it contains $\overrightarrow{f_k}$.

[2] In general, the order of evaluation literals will be considered in the global optimization phase.

Let $\hat{c}$, c_2 and c_3 be the computational costs for $\hat{f}$, f_2 and f_3, respectively. Here, $f_3 = \hat{f}$ and $c_3 = \hat{c}$. First, the total computational cost of $\sigma_{\hat{f}}R$ is $\hat{c}|R|$. As shown in Figure 2, the total computational cost for $\sigma_{(\vec{f}_2(o_1,x), \overleftrightarrow{f}_3(o_1,x))}R$ is estimated by

$$(I_{\hat{f}}(c_2 + \hat{c}) + (1 - I_{\hat{f}})(c_2 + d_2\hat{c}))|R|$$

Therefore, the radio between $\hat{c}|R|$ and the above is

$$I_{\hat{f}} + (1 - I_{\hat{f}})d_2 + \frac{c_2}{\hat{c}} = I_{f_2} + \frac{c_2}{\hat{c}}$$

If the ratio is less than 1, the optimization is profitable. Since $I_{f_2} < 1$ and $\hat{c} \gg c_2$, the above is almost always true.

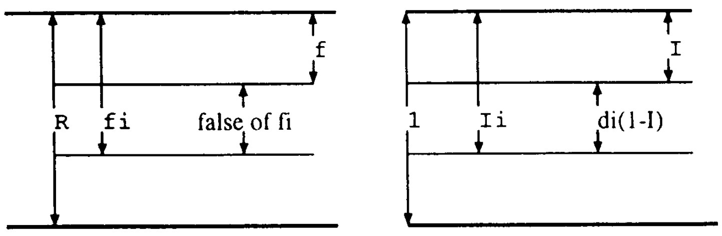

Figure 2: Selectivities and false drops

For simplicity, we use $\lambda_i = (f_1, f_2, \cdots, f_{n-1}, f_n)$ to indicate a permutation of $\lambda_i = (f_k, f_l, \cdots)$ on the condition that $f_n = \hat{f}$. The total computational cost for

$$\sigma_{(f_1 \wedge f_2 \wedge \cdots \wedge f_{n-1} \wedge f_n)}R$$

which is denoted by Cost(λ_i) or by Cost($f_1, f_2, \cdots, f_{n-1}, f_n$), is as follows:

$$Cost(\lambda_i) = I_{\hat{f}}(c_1 + c_2 + ... + c_n) + (1 - I_{\hat{f}})(c_1 + d_1c_2 + d_1d_2c_3 + ... + d_1d_2...d_{n-1}c_n) \qquad (5)$$

In [CYY$^+$92], a rank p_k is given as $p_k = c_k/(1 - d_k)$. Also, a Lemma was given to find a permutation of $(f_1, f_2, \cdots, f_{n-1}, f_n)$ for which all of the predicates are sorted in the ascending order based on their ranks, $p_1 \leq p_2 \leq p_3, \cdots$. This Lemma proves that the cost of this permutation is optimal for λ_i, if all predicates are used.

Lemma 1 *Let $f_1, ..., f_n$ and $p_1, ..., p_n$ be a permutation of $\lambda_i = (f_k, f_l, \cdots)$ and their ranks. If $p_i \leq p_j$, for any $1 \leq i < j \leq n$, then $Cost(f_1, ..., f_n)$ is the minimal cost for λ_i.*

However, the above Lemma doesn't state whether any f_i can be further pruned from the permutation, in order to reduce its computational cost.

Lemma 2 *Let $\lambda_i = (f_1, ..., f_n)$ be a permutation satisfying Lemma 1. Then $Cost(\lambda_i) \leq Cost(\lambda_i - \{f_k\})$ implies $Cost(\lambda_i - \{f_j\}) \leq Cost(\lambda_i - \{f_j\} - \{f_k\})$, for any $j \neq k$.*

The Lemma 2 states that if the elimination of f_k from λ_i increases the computational cost, then elimination of f_k from any subsequence of λ_i will increases the computational cost as well. In fact, this is used to reduce the search space of the optimization. Based on the two Lemmas, the local optimization algorithm can be obtained. The Theorem in [CYY+92] proves that the algorithm provides an optimal solution.

Algorithm (Local Optimization). Suppose that the evaluation plan of the i-th literal is $\lambda_i = (f_k, f_l, \cdots)$. Suppose their computational costs and false drops of any predicate in λ_i are known. This algorithm will generate a local optimal plan for the literal λ_i.

1. Obtain a permutation $\lambda_i = (f_1, ..., f_n)$, that satisfies Lemma 1.
2. Remove all predicates f_k in the permutation, if $d_k = 0$ and f_k is not $\hat{f}$.[3]
3. Let λ'_i be a subsequence of the permutation in which it only contains f_k if $Cost(\lambda_i - \{f_k\}) < Cost(\lambda_i)$ or f_k is $\hat{f}$. Let m be the length of the sequence. Output $Search(\lambda'_i, m)$.

The procedure $Search(\lambda, i)$ is defined as follows.

```
Search(λ, i)
        if $i = 0$ then return $\lambda$;
        let $f_i$ be the $i$-th predicate in $\lambda$;
        if $Cost(\lambda - \{f_i\}) < Cost(\lambda)$ then
            return $minCost(Search(\lambda - \{f_i\}, i - 1), Search(\lambda, i - 1))$
        else return $Search(\lambda, i - 1)$;
```

Here $minCost(\lambda_1, \lambda_2)$="if $Cost(\lambda_1) < Cost(\lambda_2)$ then λ_1 else λ_2".

3.2 Disjunctive Filtering

Let $\nu_k = (f_1, f_2, \cdots)$ be a literal, $f_1 \vee f_2 \vee \cdots$, in a conjunctive normal form. This literal $\nu_k = (f_1, f_2, \cdots)$ will be executed in the order as specified. Recall that the literal $\lambda_j = (f_1, f_2, \cdots)$ is to discard all the objects which do not satisfy any f_i in λ_j. The literal $\nu_k = (f_1, f_2, \cdots)$ is to output all the object which satisfy at least one f_i in ν_k. Therefore, the same f_i predicate will act differently depending on where it exists.

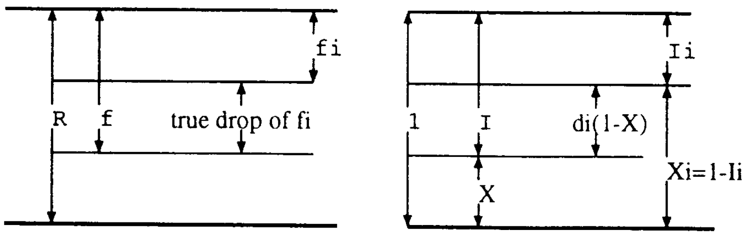

Figure 3: Selectivities, Excluding Rate and True Drops

For a f_i in ν_k, an *excluding rate*, denoted $X_{f_i} = (1 - I_{f_i})$, is defined and is illustrated as in Figure 3. If f_i is a sufficient condition of $\hat{f}$, then $I_{\hat{f}} \geq I_{f_i}$ and therefore $X_{\hat{f}} < X_{f_i}$. In comparison with the false drop, d_i, defined for a predicate f_i in a literal λ, a *true drop* of f_i, denoted x_i, is defined below.

$$x_i = (|R - \sigma_{f_i}R| - |R - \sigma_{\hat{f}}R|)/|\sigma_{\hat{f}}R|) = (X_{f_i} - X_{\hat{f}})/(1 - X_{\hat{f}})$$

[3]the false drop for $\hat{f}$ is zero.

where $X_{\hat{f}}$ and X_{f_i} are the excluding rates of $\hat{f}$ and f_i, respectively. If f_i is a sufficient condition of $\hat{f}$, then $0 < X_{\hat{f}} < X_{f_i} \leq 1$. Therefore, $0 < x_i \leq 1$.

For simplicity, we use $\nu_i = (f_1, f_2, \cdots, f_{n-1}, f_n)$ to indicate a permutation of $\nu_i = (f_k, f_l, \cdots)$. The total computational cost for

$$\sigma_{(f_1 \vee f_2 \vee \cdots, \vee f_{n-1} \vee f_n)} R$$

which is denoted by Cost(ν_i) or by Cost$(f_1, f_2, \cdots, f_{n-1}, f_n)$, is as follows:

$$Cost(\nu_i) = X_{\hat{f}}(c_1 + c_2 + ... + c_n) + (1 - X_{\hat{f}})(c_1 + x_1 c_2 + x_1 x_2 c_3 + ... + x_1 x_2 ... x_{n-1} c_n) \qquad (6)$$

In a similar fashion to the rank p_i, a new rank $p'_i = c_i/(1 - x_i)$ can be specified. Using p'_i, the same Lemmas as given in the previous section can be proved for any literal ν. Therefore, a similar Algorithm can be proved to minimize the computational cost for any ν as well.

3.3 Complexity Analysis

Some researchers like those of [HS93, CS96] define the rank by cost_per_tuple/(1-selectivity) or its inversion. It must be noted that their definition is not apropriate, at least in the case of our discussion, because such definition ignore the case *e.g.* a predicate f_2 is applied to the result of $\sigma_{f_1} R$ instead of to R itself, and so on.

The cost of optimization itself is cheap; ranks of each f_i are calculated, which cost $O(n)$. Then, f_i's are sorted according to the ascending order of their rank p_i in complexity $O(t * (n/t) \log(n/t))$, where t is the number of predicate in a conjunctive or disjunctive pharase. Hence the total cost of order f_i is $O(n \log(n/t))$. The complexity of the algorithm in either sub-section above is O(T*2^k), where T is the cost to calculate the value of the expression

$$\begin{aligned} Cost(\lambda) = {} & I * (c_1 + c_2 + ... + c_t) \\ & + (1 - I)(c_1 + d_1 c_2 + d_1 d_2 c_3 + ... + d_1 d_2 ... d_{t-1} c_t). \end{aligned}$$

For the original expensive predicate with a small number of functions in the specification, $O(T*2^k)$ is small. On the other hand, if the expensive predicate has a large number, denoted by n, of filters in the specification, the naive algorithm to find the optimal subset and order has the complexity $O(2^n * n!)$. The algorithm above reduces the cost from $O(T * 2^n * n!)$ to $O(T * 2^k)$ for some $k < n$, extending the practical applicability of our optimization scheme. Furthermore, since the ranks of expensive predicates are considered as fairly stable, optimal decomposition for a given expensive predicate does not change frequently. This means that once the optimal decomposition is derived for an expensive predicate, the DBMS can cache the optimization result.

4 General Filtering Algorithm

Now we return to the problem of optimizing (3) and (4). Let λ_i and ν_i be a result of section 3.1 and 3.2 for $\wedge f_{ij}$ and $\vee f_{ij}$, respectively. λ_i and ν_i is a local optimal evaluating plan. Then (3) and (4) is rewritten to $\vee_i \lambda_i$ and $\wedge_i \nu_i$. If we can figure out the necessary parameters computational cost and drop for λ_i and ν_i, then we are able to reuse 3.1 and 3.2 to optimize.

As shown in Figure 2, the false drop of λ_i is the product of that of f_{ij}'s, $d_{\lambda_i} = \prod_j d_{ij}$. Denoting $\prod_{j=1}^{0} d_{ij} \equiv 1$, then similar to (5), the computational cost of of each object, λ_i, is given by

$$c_{\lambda_i} \;=\; I * (c_{i1} + c_{i2} + ... + c_{it_i}) + (1 - I)(c_{i1} + d_{i1} c_{i2} + d_{i1} d_{i2} c_{i3} + ... + d_{i1} d_{i2} ... d_{i,t-1} c_{it_i})$$

$$= I * \sum_{j=1}^{t_i} c_{ij} + (1-I) \sum_{k=1}^{t_i} c_{ik} \prod_{j=1}^{k-1} d_{ij}$$

Therefore, the rank of λ_i can be calculated by

$$p_{\lambda_i} = c_{\lambda_i}/(1 - d_{\lambda_i})$$

Now we can reuse the conclusion in sectinos 3.1 and give the general filtering algorithm for (3):

Algorithm (General Filtering)

1. Optimize each pharse as in the section 3.1. Let the results be λ_i for each phares i.
2. Sort λ_i's in the ascending order of their ranks p_{λ_i}.
3. Using the algorithm in section 3.2 to prune some λ_i's.

The optimization of (4) is in the simillar way. Referring to Figure 3 and equation (??), we have $d_{\nu_i} = \prod_j d_{ij}$,

$$c_{\nu_i} = X * \sum_{j=1}^{t_i} c_{ij} + (1-X) \sum_{k=1}^{t_i} c_{ik} \prod_{j=1}^{k-1} d_{ij}$$

and

$$p_{\nu_i} = c_{\nu_i}/(1 - d_{\nu_i})$$

Our optimization is excellent in that it is

Concise. The algorithm in section 3.1 is the only one needed to be implemented.

Simple. The only information about f_{ij}'s is their cost and selectivity. Other paprameters are simply derived.

The only semantics of f_{ij} is its dependency (necessary or sufficient condition) against f. We avoid to discuss the semantics between f_{ij}'s because it makes the optimization impossible, though theoretically, avoiding of semantics sacrifies the optimality. The complexity of the general filtering is twice as that of the algorithm in local optimizaiton, plus the calculation of c_{λ_i} and c_{ν_i} for each i. The costs of calculating d_{λ_i}, d_{ν_i}, p_{λ_i} and p_{ν_i} are negligible.

5 Conclusion

We have proposed and described a new scheme of optimizing queries including expensive predicates, based on the concept of filtering. Optimization algorithms are proposed to find the optimal plan of evaluating expensive functions. The analytical study based on the model shows the advantage of our method. The experimental study for the geographical database shows the feasibility of this method.

New methods and extension of the cost model poses the following open problems; 1) For an actual user defined data type, the rank of each expensive predicate should varies depending on the sizes the instances. 2) Since the efficiency of the algorithms depends on the specification of the expensive predicate, semantic information associated with these predicates is necessary.

References

[Atk87] M. P. Atkinson and O. P. Buneman: Types and Persistence in Database Programming Languages, *ACM Computing Surveys*, Vol.19, No.2, pp.105-190, June 1987.

[CYY+92] H. Chen, X. Yu, K. Yamaguchi, H. Kitagawa, N. Ohbo & Y. Fujiwara: Decomposition - An Approach for Optimizing Queries Including ADT Functions, *Information Processing Letters*, Vol. 43(6), pp.327-333, 1992.

[CS93] S. Chaudhuri & K. Shim: Query Optimization in the Presence of Foreign Functions. *Proc. VLDB'93*, pp.529-542, Dublin, August 1993.

[CS96] S. Chaudhuri & K. Shim: Optimization of Queries with User-defined Predicates. *Proc. VLDB'96*. pp.87-98, Bombay, India, 1996.

[Gutt84] Antonin Guttman: "R-Trees: A Dynamic Index Structure for Spatial Searching", *13th ACM-SIGMOD*, Boston, New York, pp. 47-57.

[Haa89] L. M. Haas, et al.: Extensible Query Processing in Starburst, *Proc. of ACM SIGMOD'89*, pp.377-388, Portland, Oregon, 1989.

[HS93] J. M. Hellerstein & M. Stonebraker: Predicate migration: Optimization queries with expensive predicate. *Proc. ACM-SIGMODE'93*, pp.325-335, Minneapolis, MN, May 1994.

[Hel94] J. M. Hellerstein: Practical Predicate Placement. *Proc. ACM-SIGMODE'94*, pp.325-335, Minneapolis, MN, May 1994.

[KBZ86] R. Krishnamurthy, H. Boral, C. Zaniolo: Optimization of Nonrecursive Queries, *Proc. of VLDB'86*, pp.128-137, Kyoto, 1986.

[PHH92] H. Pirahesh, J.M. Hellerstein, & W. Hasan: Extensible/Rule-Based Query Rewrite Optimization in Starburst. *Proc. ACM-SIGMOD'92*, pp.39-48, San Diego, June 1992.

[PTSE95] D. Papadias, Y. Theodoridis, T. Sellis & M. J. Egenhofer: Topological Relations in the World of Minimum Bounding Rectangles: A Study with R-trees. *Proc. ACM-SIGMOD'95*. pp.92-103, San Jose, Ca., May 1995.

[Sto88] M. Stonebraker: Inclusion of New Types in Relational Data Base Systems, *Readings in Database Systems*, edited by M. Stonebraker, Morgan Kaufman, pp.480-487, 1988.

[Worb95] Michael F. Worboys: "GIS: A Computing Perspective", Taylor & Francis, 1995.

[YKY+91] K. Yajima, H. Kitagawa, K. Yamaguchi, N. Ohbo & Y. Fujiwara: Optimizatino of Queries Including ADT Functions, *Proc. DASFAA'91*, pp.366-373, Tokyo, April 1991.