ELSEVIER

The Journal of Systems and Software 62 (2002) 161–179

The Journal of Systems and Software

www.elsevier.com/locate/jss

Heuristic approach for early separated filter and refinement strategy in spatial query optimization

Ho-Hyun Park, Hyung-Ju Cho, Chin-Wan Chung *

Department of Electrical Engineering and Computer Science, Division of Computer Science, Korea Advanced Institute of Science and Technology, 373-1 Yusong-ku, Kusong-dong, Taejon 305-701, South Korea

Received 20 November 2000; received in revised form 2 April 2001; accepted 19 June 2001

Abstract

Recently, we proposed an optimization strategy for spatial and non-spatial mixed queries. In the strategy, the filter step and the refinement step of a spatial operator are regarded as individual algebraic operators, and are early separated at the algebraic level by the query optimizer. By doing so, the optimizer using the strategy could generate more diverse and efficient plans than the traditional optimizer. We called this optimization strategy the *Early Separated Filter And Refinement* (ESFAR).

In this paper, we improved the cost model of the ESFAR optimizer considering the real life environment such as the LRU buffer, the clustering of the dataset, and the selectivity of the real data distribution. And we conducted a new experiment for ESFAR by comparing the optimization result generated by the new cost model and the actual execution result using real data. The experimental result showed that our cost model is accurate and our ESFAR optimizer estimates the costs of execution plans well.

Since the ESFAR strategy has more operators and more rules than the traditional one, it consumes more optimization time. In this paper, we apply two existing heuristic algorithms, the *iterative improvement* (II) and the *simulated annealing* (SA), to the ESFAR optimizer. Additionally we propose a new heuristic algorithm to find a good initial state of II and SA. Through experiments, we show that the II and SA algorithms in the ESFAR strategy find a good sub-optimal plan in reasonable time. Mostly the heuristic algorithms find a lower cost plan in less time than the optimal plan generated by the traditional optimizer. Especially the II algorithm with the initial state heuristic rapidly finds a plan of a high quality.

1. Introduction

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* (Orenstein, 1986). 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 (Aref and Samet, 1991; Becker and Güting, 1992; Ooi et al., 1989) 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 algebraic operator. However, when spatial predicates and non-spatial predicates are mixed in a query and if spatial indexes exist on a class referenced by spatial predicates, 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.

1.1. Related work

Several spatial database systems have addressed the optimization problem for the spatial and non-spatial mixed query in the literature (Samet and Aref, 1995). GEOQL's optimizer decomposes a spatial query into spatial subqueries and non-spatial subqueries (Ooi et al., 1989). 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

* Corresponding author. Tel.: +82-42-869-3537; fax: +82-42-869-3510.

E-mail addresses: hhpark@islab.kaist.ac.kr (H.-H. Park), hjcho@islab.kaist.ac.kr (H.-J. Cho), chungcw@islab.kaist.ac.kr (C.-W. Chung).

0164-1212/01/$ - see front matter
PII: S0164-1212(01)00138-8

by the spatial query 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 (Becker and Güting, 1992) performs a rule-based optimization using an algebra which is called the geo-relational algebra. GRAL's optimizer uses a pre-defined partial order between some algebraic operators to find a good execution order in a heuristic manner. SAND's optimizer (Aref and Samet, 1991) 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 optimization techniques for both spatial and non-spatial operations (Patel et al., 1997). Paradise's query optimizer is written by an optimizer generator called OPT++ (Kabra and DeWitt, 1996). However, none of the above optimizers considers the filter and refinement steps of spatial operators separately as individual operators. Therefore, the above optimizers cannot generate the plans which will be presented throughout this paper.

1.2. Summary of our work

Our recent research (Park et al., 1999, 2000) presented query optimization strategies which addressed this problem. The first strategy 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. As the second strategy, several refinement operations can be combined, and as the third strategy several filter operations can also be combined. We called the optimization technique utilizing these strategies the *Early Separated Filter And Refinement* (ESFAR).

This paper also presented a rule-based optimization technique for ESFAR. The input query of a rule-based optimizer is in an algebraic form. In this paper, we used the *Spatial Object Algebra* (SOA) (Park et al., 1997) to represent the input query of our optimizer. In addition, we needed a new object algebra for ESFAR which separates the operators in SOA into filter step operators and refinement step operators. We defined the *Intermediate Spatial Object Algebra* (ISOA) as the new object algebra. Using ISOA, we derived some optimization rules for ESFAR. In this paper, we implemented the execution algorithms for ISOA. Through experiments using the TIGER data (US Bureau of the Census, 1995), we showed that the ESFAR optimization strategy generates more efficient query execution plans than the traditional one in many cases.

In this paper, we improved the cost model of the ESFAR optimizer, and implemented the ESFAR optimizer using the Volcano optimizer generator (VOG) (Graefe and McKenna, 1993), and ran the optimizer for the TIGER data, and compared the optimization results and the actual execution results. Especially our improved cost model considered the real life environment such as the LRU buffer, the clustering of the data, and the selectivity considering the real data distribution while our old cost model (Park et al., 1999) assumed the uniform distribution of spatial and non-spatial objects, and no buffering and no clustering. And we conducted a new experiment for ESFAR by comparing the optimization result generated by the new cost model and the actual execution result using the TIGER data. According to the experimental result, the optimization results were similar to the actual execution results in most cases. The experimental result showed that our cost model is accurate and our optimizer estimates the lowest cost execution plan well.

Actually the ESFAR optimization technique always generates more efficient execution plans than the traditional optimization technique because the ISOA operators include the traditional operators and the ESFAR rules include the traditional optimization rules. However, because of more operators and more rules, the ESFAR optimizer consumes more optimization time than the traditional optimizer. In this paper, we apply heuristic algorithms to the ESFAR optimizer.

VOG internally has the exhaustive search algorithm to generate the optimal plan. We replaced the exhaustive search algorithm with heuristic search algorithms. As the heuristic search algorithms, we implemented the two representative hill climbing algorithms (Ioannidis and Kang, 1990), the *iterative improvement* (II) and the *simulated annealing* (SA). II and SA randomly choose the initial state. However, in the hill climbing algorithm, a well chosen initial state may improve both the optimization time and the optimization quality significantly. In this paper, we propose a new algorithm for selecting an initial state of II and SA considering the characteristics of the ESFAR strategy.

Through experiments, we show that the II and SA algorithms in the ESFAR strategy find good sub-optimal plans in reasonable time. Mostly the heuristic algorithms find plans whose costs are lower than that of the optimal plan generated by the traditional optimizer. Especially the II algorithm with the initial state heuristic rapidly finds a plan of a high quality.

1.3. Paper organization

The remainder of this paper is organized as follows. In Section 2, we describe ESFAR which is regarded as the background for this work. Sections 2.1–2.3 summarize our previous work and Sections 2.4 and 2.5 contain our new contributions which are the improved cost model and a new experiment. In Section 3, we apply two hill climbing algorithms, II and SA, to the ESFAR strategy and propose a new algorithm for finding a good

initial state. The experimental results are presented in Section 4. In Section 5, we conclude this paper.

2. ESFAR

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.
2. We consider only the R*-tree (Beckmann et al., 1990; Brinkhoff et al., 1993) as a spatial indexing and the B+-tree as a non-spatial indexing because they are the most general indexing methods in spatial data and non-spatial data, respectively.

2.1. Strategy

We illustrate our optimization strategy using an example. Suppose that a mixed query which consists of a spatial select operation (S_SEL) and a non-spatial select operation (N_SEL) was issued by the user. If the separation of filter and refinement of the S_SEL operation is possible at the algebraic operator level of the query optimizer and an R*-tree exists for the spatial attribute, the S_SEL operation can be separated into the spatial select filter (SSF) and the spatial select refinement (SSR) operations. Obviously, SSR is a SEL operation of the relational algebra because it is generated from a select operation (S_SEL). Therefore, the above query can be transformed into a query which is in the order of "SSF – N_SEL – SSR" by the select commutative rule of the relational algebra. The processing of the original query in the order of "SSF – N_SEL – SSR" can be more efficient than the order of "N_SEL – S_SEL" or "S_SEL – N_SEL." If the filter and refinement steps of S_SEL are not separated at the algebraic operator level, the plan like "SSF – N_SEL – SSR" cannot be generated, but only the plan like "N_SEL – S_SEL" or "S_SEL – N_SEL" can be generated.

The spatial join operation (S_JOIN) 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: $\mathrm{JOIN}\theta_1 \wedge \theta_2(R,S) = \mathrm{SEL}\ \theta_1(\mathrm{JOIN}\theta_2(R,S))$.). Therefore, all relational algebra rules related to SEL can also be applied to SJR. One important thing to be considered about a spatial predicate is that the filter step predicate is different from the actual predicate as pointed out in (Papadias et al., 1995).

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.

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 paragraphs, 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 (Silberschatz et al., 1997), the refinement operation can be combined with other non-spatial select operations to generate another SEL operation. 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.

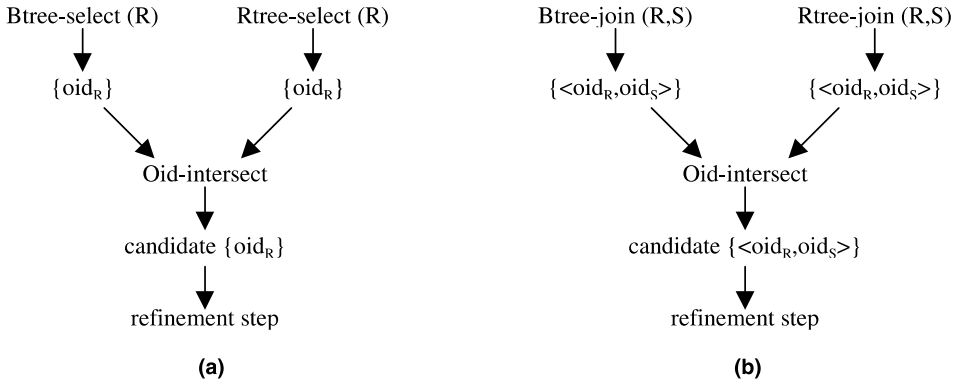

Fig. 1. Oid-intersection between spatial and non-spatial operations.

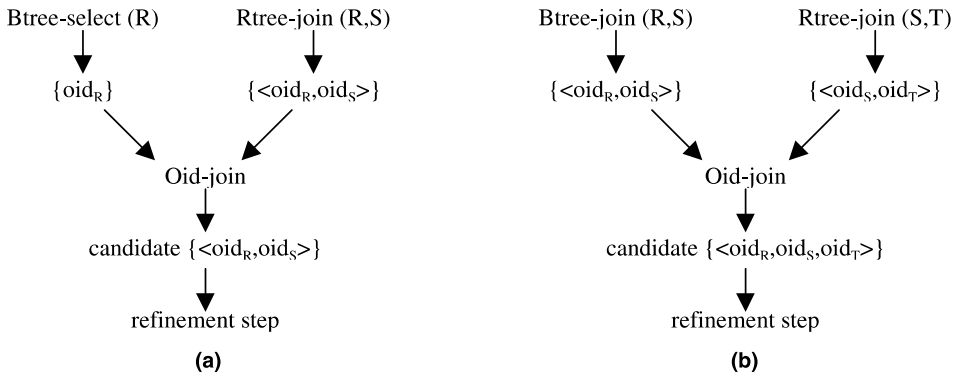

Fig. 2. Oid-join between spatial and non-spatial operations.

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.

As 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 (Mohan et al., 1990) and the *Oid-join* technique (Blasgen and Eswaran, 1977; Cho et al., 1997; Valduriez, 1987) to spatial and non-spatial mixed query processing.

Figs. 1 and 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 get 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. The 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, [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. 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 large effect in the spatial and non-spatial mixed query processing.

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

2.2. Algebra

In Section 2, we separated some algebraic operators of the SOA into those of filter and refinement steps. Non-spatial objects can also be processed in two steps like spatial objects. As the first step, if B+-trees was already built on the classes which are referenced in the non-spatial predicates of the query, we can get 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 were obtained from 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 were obtained from the filter step.

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 separations are only for the purpose of optimization. We call such an intermediate form of 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 the case of the SOA (Park et al., 1997), 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 2.1, all the refinement operations correspond to the SEL operation of the relational algebra.

2.3. Rules

In this section, we present optimization rules for Strategies 1 and 3 using the ISOA. As we mentioned in Section 2.1, 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.

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)))

In Figs. 1 and 2, we saw the combined filtering examples only for two predicates. The combined filtering between two predicates may be simple because such a combining is only an 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 predicate. Then, the predicates in the SOA expression can be evaluated in an order like that of the tree traversal and the combined filtering between them can be represented by the sequence of Oid-joins. In that case, 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))))

We will give an example to show the application of the above rules to a complex query. Consider the following OQL query:

OQL 1. `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. Eq. (1) is an SOA-expression for OQL 1.

$$S\_\text{JOIN}\ \theta_2\ (S\_\text{JOIN}\ \theta_1\ (N\_\text{SEL}\ \theta_3\ (a),\ b),\ S\_\text{SEL}\ \theta_4\ (c)). \quad (1)$$

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

$$\text{SEL}\ \theta_3 \wedge \theta_4\ (S\_\text{JOIN}\ \theta_2\ (S\_\text{JOIN}\ \theta_1\ (\text{NSF}\theta_3\ (a), b), \text{SSF}\theta_4\ (c))). \quad (2)$$

By Rule 4, the select-join commutative rule and the select-merge rule

$$\text{SEL}\theta_3 \wedge \theta_4 \wedge \theta_1\ (S\_\text{JOIN}\ \theta_2\ (\text{OJ}\ (\text{NSF}\ \theta_3\ (a), \text{SJF}\ \theta_1\ (a, b)), \text{SSF}\ \theta_4\ (c))). \quad (3)$$

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

$$\text{SEL}\theta_3 \wedge \theta_4 \wedge \theta_1 \wedge \theta_2\ (\text{OJ}\ (\text{OJ}\ (\text{NSF}\theta_3\ (a), \text{SJF}\theta_1\ (a, b)), \text{OJ}(\text{SJF}\theta_2\ (a, c), \text{SSF}\theta_4\ (c)))). \quad (4)$$

2.4. Cost model

In our previous work (Park et al., 1999), we used a simple cost model which assumed no buffering and no clustering. In a real life environment, however, most database management systems have the LRU buffer. Therefore, we newly apply the LRU buffered cost model to this implementation and the experiment of the ESFAR optimizer. We use the TIGER data (US Bureau of the Census, 1995) as our experimental data. Since the TIGER data is clustered on a non-spatial attribute, we apply the clustering cost model for non-spatial operations. We also observed that our experimental data is highly clustered on the spatial attribute. Therefore we also apply the clustering cost model for spatial

Table 1
The statistics of the TIGER data

County	# of Objectives		Average # of points		MBR density		Domain area (10^5)
	A	H	A	H	A	H	
Kern	113 407	26 014	3.7	8.9	0.26	0.32	$26 * 10$
Riverside	91 751	15 440	3.5	12.2	0.21	0.52	$32 * 7$
San Bernadino	146 026	28 187	3.5	12.4	0.22	0.40	$37 * 19$

operations. The cost model used in these experiments are summarized in Appendix A. [2] The TIGER data are skewed for both spatial and non-spatial attributes. To estimate the non-spatial selectivity well, we used the equi-depth histogram (Piatetsky-Shapiro and Connell, 1984) whose number of buckets is 20. And for the spatial selectivity, we used the equi-area histogram (Acharya et al., 1999) whose number of buckets is $20 * 10$.

2.5. Experimental results

To measure the performance of the ESFAR optimization strategies, we implemented an optimizer using the VOG (Graefe and McKenna, 1993) and execution algorithms of the ISOA operators. 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 buffer entries were fixed to 4 KB and 256, respectively. The experimental data is the TIGER/Line data (US Bureau of the Census, 1995) extracted from roads and hydrographies of three counties of the California state. Our TIGER data is sorted along the TLID (TIGER/Line IDentification number) field. We built the B+-tree on the TLID field 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 1. A and H, respectively, represent CFCC (Census Feature Class Code) (US Bureau of the Census, 1995) for roads and hydrographies in the TIGER data in Table 1.

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". The experiments were performed in two stages. At the first stage, we measured the actual execution time of some possible plans which can be generated by a traditional optimizer and the ESFAR optimizer. At the second stage, we ran the two optimizers and compared the optimization results and the actual execution results. The first stage was performed in our previous research (Park et al., 2000). Now we conduct experiments for the second stage. This section includes the summary of the experimental results. [3] Tables 2 and 3 show one of the experimental results of each stage, respectively. To see the detailed results, refer to Park (2001).

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 2.3 is more expensive than that only by the traditional rules, the ESFAR optimizer chooses the execution plan by the traditional rules.

The first experiment was performed for the following "non-spatial select and spatial select" type of query:

OQL 2. `select a from a in roads where a.LINE` *s_intersect* `s_polygon(`$x_1, y_1, \ldots, x_n, y_n$`) and a.TLID >` `z;`

In the above query, `s_polygon(`$x_1, y_1, \ldots, x_n, y_n$`)` 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: 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)

The selectivity (S_{nsp}) of the non-spatial select is controlled by the integer constant `z` and the selectivity (S_{sp})

[2] In this paper, we do not develop a new cost model but newly apply existing cost models to the clustered and buffered environment. Therefore, we summarize the cost model in Appendix A rather than presenting it in this section.

[3] We describe the experimental result of the first stage in more detail because the experimental result is considered as the basis of the heuristic algorithm of the next section.

Table 2
Actual 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

S_{sp}	z	TRA1	TRA2	ESFAR1	ESFAR2	ESFAR3
	1	3296	3789	2989	2593	*2087*
0.20	5	3917	6015	5745	*3778*	4231
	25	*7367*	14 144	19 384	7622	13 773
	1	816	3373	788	*661*	1284
0.05	5	*961*	4845	1527	978	2054
	25	*1650*	10 330	4900	1796	8806
	1	170	3258	150	*137*	1005
0.01	5	200	4511	279	*199*	1965
	25	*330*	9431	849	352	7556

Table 3
Optimization results for the query of "spatial select and non-spatial join" type for Riverside county when the spatial select is performed on hydrography data

S_{nsp}	z	Traditional optimizer			ESFAR optimizer		
		Plan	Cost	Res	Plan	Cost	Res
	1	TRA1	3787	O	ESFAR3	2131	O
0.20	5	TRA1	4463	O	TRA1	4463	X
	25	TRA1	6997	O	TRA1	6997	O
	1	TRA1	918	O	ESFAR2	750	O
0.05	5	TRA1	1082	O	TRA1	1082	O
	25	TRA1	1696	O	TRA1	1696	O
	1	TRA1	207	O	ESFAR2	171	O
0.01	5	TRA1	243	O	TRA1	243	X
	25	TRA1	380	O	TRA1	380	O

of the spatial select by the size of the constant polygon `s_polygon`$(x_1, y_1, \ldots, x_n, y_n)$. 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 lower than S_{sp}, executing the non-spatial operation first (TRA1) shows the best performance.

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

OQL 3. `select a, h from a in roads, h in hydrographies where a.LINE` *s_intersect* `s_polygon`$(x_1, y_1, \ldots, x_n, y_n)$ `and -z` $\leqslant$ `a.TLID-b.TLID` $\leqslant$ `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

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)

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. If the spatial selectivity is low ($S_{sp} = 0.01$) or the non-spatial join selectivity is high ($z = 25$), executing spatial selection first (TRA1) performs best in many cases because the result size from the spatial select is much smaller than the join size.

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

OQL 4. `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. 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 (Strategies 2 and Strategy 3)

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. In many cases, ESFAR3 has the faster response time than TRA. This is because the number of refinement operations in ESFAR3 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 average hit ratio by the spatial join refinement was observed to be approximately 25%. Sometimes, ESFAR3 performs worse than TRA. This depends on the number of oid-tuples resulting from the combined filtering. If the intermediate result size from the combined filtering is large, there is a large overhead for reading oid-tuples and the relavant objects, therefore ESFAR3 performs worse than TRA.

For the above three types of queries, our optimizer estimates the costs of the execution plans and the lowest cost plan well. Wrong optimization results occur only when the difference between the cheapest execution plan and the second plan is small. Table 3 shows an optimization result for a query of "spatial select and non-spatial join" type for Riverside county while Table 2 shows the actual response time for the query. In the cost model of the spatial join, through trial and error, we set the N^* value of Eq. (A.31) to 100. And we set the c value of Eq. (A.40) to 3.3 as in the case of the original paper (Huang et al., 1997) because we use the similar datasets as the paper did.

3. Heuristic approach

Until now, we used the exhaustive search strategy by the dynamic programming built in the VOG as an optimization method. In the traditional select-join query, the query optimization problem for the optimal solution is known to be NP-complete (Selinger et al., 1979; Silberschatz et al., 1997; Ullman, 1988). Since the ESFAR optimizer has more algebraic operators and rules than the traditional optimizer, the ESFAR optimization problem is more complex than the traditional one. Therefore, as the query becomes complex, a heuristic algorithm for the ESFAR optimization is essential. In this section, we present a heuristic approach for ESFAR.

3.1. Randomized algorithms for query optimization

Since finding the optimal solution for a complex query is very time consuming, or nearly impossible because of the space complexity, many heuristic algorithms have been developed. Among them, the randomized algorithms (Ioannidis and Kang, 1990) such as the II and the SA are best known.

In the query optimization problem, the whole search space can be modeled as an undirected graph, each node of which corresponds to an execution plan. A node in the graph is called a *state* in the space and an edge corresponds to a *move*. Randomized algorithms perform random walks between states until they find a suboptimal plan. A move is called *downhill* if it leads to a lower cost plan, or *uphill* if it leads to a higher cost plan. The randomized algorithm is also called the hill climbing algorithm.

II continuously applies the following local optimization: The local optimization algorithm starts with randomly choosing an initial state in the space. Then it finds a random neighbor by applying a possible transformation from the plan of the current state. If the move to the random neighbor is downhill, the algorithm accepts the move, else it finds another neighbor. The above random move continues until there is no neighbor with lower cost (i.e., it reaches a *local minimum*). If the stopping condition is met, II terminates with the minimum of the local minimum costs. Generally the stopping condition is set to the number of iterations for the local optimizations.

```
  Algorithm II( )
(1)     minS = S∞;
(2)     WHILE NOT (stopping_condition) DO
(3)       S = random state;
(4)       WHILE NOT (local_minimum(S)) DO
(5)         S′ = random state in neighbors(S);
(6)         IF cost (S′) < cost(S) THEN
          S = S′;
(7)       END WHILE;
(8)       IF cost(S) < cost(minS) THEN
          minS = S;
(9)     END WHILE;
(10)    return(minS);
  END II
```

While II accepts only downhill moves, SA accepts uphill with some probability as well as downhill, trying to avoid a high cost local minimum. As in the case of II, the SA algorithm consists of two loops. The inner loop of SA is called a *stage*. Each stage works with a temperature T which determines the probability of the uphill move. T (consequently the probability of the uphill move) gradually decreases as the stage number increases. Each stage ends when it reaches an *equilibrium*. An equilibrium is usually determined by a fixed number of iterations of the inner loop. SA stops when T is near to zero, i.e., the algorithm reaches the *frozen* state.

```
  Algorithm SA( )
(1)     S = random state;
(2)     T = T0;
(3)     minS = S;
(4)     WHILE NOT (frozen) DO
(5)         WHILE NOT (equilibrium) DO
(6)           S' = random state in neigh-
        bors(S);
(7)           ΔC = cost(S') - cost(S);
(8)           IF (ΔC ⩽ 0) THEN S = S';
(9)           IF (ΔC > 0) THEN S = S' with
        probability e^(-ΔC/T);
(10)          IF cost(S) < cost(minS) THEN
        minS = S;
(11)        END WHILE;
(12)        T = reduce(T);
(13)    END WHILE;
(14)    return(minS);
  END SA
```

3.2. Selection of an initial state

The two algorithms of the previous section randomly chose the initial state. In the hill climbing algorithm, a well chosen initial state improves both the optimization time and the optimization quality significantly. In the experiments of Section 2.5, we showed some characteristics of the performance of the ESFAR strategies for simple queries. The efficiency of the ESFAR strategies is primarily influenced by the selectivities of the spatial and non-spatial selects and joins. First, only for a spatial select and a non-spatial select, if the selectivity of the non-spatial predicate is higher than that of the spatial predicate or both are similar, ESFAR performs better. Otherwise, i.e., if the selectivity of the non-spatial predicate is lower than that of the spatial predicate, the traditional query evaluation method evaluating the non-spatial predicate first performs better. Second, for a spatial select and a general join, [4] if the result size of the spatial selection predicate is large and the selectivity of the general join is low, ESFAR performs better. Otherwise, the traditional approach evaluating the selection predicate first performs better. In this section, we applied the results of Section 2.5 to complex select-join queries for selecting an initial state of the II and SA heuristics.

For select-join predicates, the traditional optimizer usually applied the "selection as early as possible" heuristic (Selinger et al., 1979; Silberschatz et al., 1997; Ullman, 1988). In a spatial and non-spatial mixed query, however, it is no longer a good heuristic because the evaluation cost of a spatial selection predicate can be higher than that of a non-spatial join predicate in some cases. We evaluate the non-spatial selection predicates and the filter steps of spatial selection predicates as early as possible against the refinement steps of spatial selection predicates and general join predicates because the non-spatial selection predicates or the filter steps of spatial selection predicates are generally cheaper than the refinement steps of spatial selection predicates and general join predicates.

First, we consider the selection predicates. For each class, if there are one or more selection predicates, we sort the selection predicates by the following order: (1) index predicates; (2) remaining non-spatial predicates; (3) all spatial predicates. The reason behind this ordering is that the index search is generally faster than the sequential search, and for the sequential search the non-spatial predicate evaluation is much cheaper than the spatial predicate evaluation. If a spatial predicate has an index, the predicate is divided into (1) index predicates and (3) all spatial predicates because the index predicate is evaluated only by the filter step and it must be evaluated again by the refinement step. In the above ordering, since the index probing for the spatial predicate involves only the filter step, if the selectivities between spatial and non-spatial predicates are similar, we assume their index probing costs are also similar.

After the predicate ordering, we choose the most selective two index predicates. Let the two predicates be p_1 and p_2, and their selectivities be S_{p_1} and S_{p_2} ($S_{p_1} \leqslant S_{p_2}$). If there is no index predicate or only one index predicate, S_{p_1} and/or S_{p_2} are assigned 1 because the whole search must be performed when there is no index for a predicate. The reason we choose the most two selective predicates is for the possibility of the combined filtering strategy using the index intersection. However, if the index intersections are applied for too many predicates, the performance may decrease. Therefore, we choose maximally two predicates.

Based on the results of Section 2.5, we applied the following heuristic rule for the selection predicates. If S_{p_1} is low or the difference between S_{p_1} and S_{p_2} is large, perform SF [5] only for p_1, else if the selectivity difference

[4] The general join means the spatial or non-spatial join.

[5] The SF operator was defined in Section 2.2.

is small and S_{p_2} is not equal to 1, perform the combined filtering for p_1 and p_2 (i.e., SF for p_1 – SF for p_2 – OJ). After evaluating index predicates, we evaluate all unevaluated non-spatial predicates if any according to the "selection as early as possible" heuristic rule. Finally for all spatial predicates, if the estimated result size for all selection predicates is small, we evaluate the spatial predicates, otherwise we defer the spatial predicates evaluation until join predicates processing.

After selection predicates processing, we sort join predicates in an increasing order by the estimated result sizes. If the input to a join is the result of another operation, only the evaluated result until now is reflected on the input to the join, i.e., the deferred evaluation is not reflected on the input to the join. The following is the heuristic rule between spatial selection refinements and/or spatial join refinements and a general join based on the results of Section 2.5. For each join predicate q_j, if S_{q_j} is low and there are unevaluated spatial predicates on child nodes, evaluating the non-spatial join or the filter step of the spatial join first may be cheaper than evaluating the child nodes' spatial predicates first. Therefore, in this case, we pull up the unevaluated child nodes' predicates over the non-spatial join or the filter step of the spatial join.

Generally, the result of index operations such as the *B-tree select*, *B-tree join*, *R-tree select* and *R-tree join* is an Oid-tuple collection and these index operations correspond to the filter step of the operations. When the filter step of the join predicate (JF) is evaluated, [6] if OJ is possible, i.e., if the evaluated results of all child nodes are Oid-tuple collections, we perform the combined filtering through OJ between the child nodes' results and the result of JF. By this combined filtering, we can perform JF by only using indexes in a complex query and we can save the object fetch cost. If OJ is not possible, we must perform JF with actual object fetches.

Meanwhile, if S_{q_j} is high, we evaluate the unevaluated child nodes' predicates before the join evaluation because evaluating the non-spatial join or the filter step of the spatial join first may be more expensive than evaluating the child nodes' predicates first.

If q_j is a spatial predicate, we defer the evaluation of the predicate along with the pulled-up spatial predicates from the child nodes until the processing of the next join predicate.

In this way, if we have processed all join nodes, we finally evaluate all unevaluated spatial predicates after the last join. Our heuristic algorithm is shown below:

```
initial_state (Query Q, Classes R[])
(1)   FOR each class R_i ∈ R[] DO
(2)     sort selection predicates by the
      following order:
(3)       index predicates – remaining
      non-spatial predicates – all spa-
      tial predicates;
(4)     p1 = the most selective index
      predicate if any;
          /* if not, assume S_p1 = 1 */
(5)     p2 = the next selective index
      predicate if any;
          /* if not, assume S_p2 = 1 */
(6)     IF S_p1 is low OR S_p1 ≪ S_p2 THEN
(7)       SF for p1;
(8)     ELSE IF S_p1 ≈ S_p2 AND S_p2 ≠ 1 THEN
(9)       SF for p1 – SF for p2 – OJ;
(10)    evaluate all unevaluated non-
      spatial predicates if any;
(11)    IF the estimated result size for
      all selection predicates is small
      THEN
(12)      evaluate all spatial predi-
      cates if any;
(13)    ELSE
(14)      defer evaluation of all spa-
      tial predicates if any;
(15)  END FOR;
(16)  sort join predicates in an in-
      creasing order by estimated re-
      sult sizes;
(17)  FOR each join predicate q_j ∈ Q DO
(18)    IF S_qj is low THEN
(19)      pull up the unevaluated child
      nodes' predicates if any;
(20)      IF OJ possible THEN
(21)        JF for q_j;
(22)        OJ with child nodes' re-
      sults;
(23)      ELSE
(24)        JF for q_j; /* for non-spatial
      join, JF == JOIN */
(25)    ELSE
(26)      evaluate the unevaluated
      child nodes' predicates if any;
(27)      JF for q_j; /* for non-spatial
      join, JF == JOIN */
(28)    defer evaluation of all uneval-
      uated spatial predicates if any;
(29)  END FOR;
(30)  evaluate all unevaluated spatial
      predicates if any;
END initial_state
```

[6] For the non-spatial join, the filter step itself is the join.

Table 4
Parameters of experimental data and computation

Parameters	Descriptions	Values
$\|\|R\|\|$	Number of objects in class R[a]	100,000
D^2	Area of total space	$10^5 * 10^5$
s_R	Size of non-spatial parts of an object in R	50 bytes
Page_Size	Page size	4096 bytes
LRU buffer	Number of LRU buffer entries	256
Point_Size	Size of a point	8 bytes
DA	Disk access time	10 ms
$T_{\text{edge–rect}}$	CPU time for edge–rectangle test	40 μs
$T_{\text{edge–edge}}$	CPU time for edge–edge test	20 μs
hit_S	Hit ratio for spatial selection	60%
hit_J	Hit ratio for spatial join	25%

[a] Contains both non-spatial part and spatial part.

4. Experiments

We implemented the two representative randomized search algorithms, II and SA, and added both algorithms as the search engine of VOG. Additionally we implemented our heuristic algorithm `initial_state` for finding a good initial state for II and SA, respectively. We call the two randomized algorithms with our initial state heuristic II_{is} and SA_{is}, respectively. The experimental environment such as the platform, the memory size, the page size and the LRU buffer is the same as Section 2.5. In these experiments, we assume that both spatial and non-spatial data are synthetically generated which are uniformly distributed. We used the same cost model as Section 2.5 which is summarized in Appendix A. The parameters of experimental data and computation are summarized in Table 4.

The query types used in these experiments are as follows:

(1) Q1: n-way non-spatial join with one spatial selection per class,
(2) Q2: n-way spatial join with one spatial selection per class,
(3) Q3: n-way mixed joins with n mixed selections.

In Q3, each join is randomly selected between the spatial join and the non-spatial join, and each selection is also randomly selected between the spatial selection and the non-spatial selection and randomly distributed over n classes. For these types of queries, we ran the following search algorithms:

(1) OPT_{TRA}: the exhaustive algorithm built-in VOG using the traditional rules for finding the optimal plan,
(2) $\text{OPT}_{\text{ESFAR}}$: the exhaustive algorithm built-in VOG using the traditional and ESFAR rules for finding the optimal plan,
(3) II: the II algorithm using the traditional and ESFAR rules,
(4) SA: the SA algorithm using the traditional and ESFAR rules,
(5) II_{is}: the II algorithm using the traditional and ESFAR rules with the initial state heuristic,
(6) SA_{is}: the SA algorithm using the traditional and ESFAR rules with the initial state heuristic.

In the above algorithms, the initial states of II and SA are randomly generated. The parameters of II and SA are summarized in Table 5. In Algorithm II, the initial states are randomly generated for every local optimization. This is done by a number of random transformations ($5 * n$ in our experiments) regardless of the cost from the previous local minimum plan. In `initial_state` heuristic, we set the criteria of the low selectivity for the select operator to 1/20. If the total selectivity for select operators per class is lower than 1/20, we evelauated all refinement operations, otherwise, we deferred the evaluation. If the difference between S_{p_1} and S_{p_2} is greater than 0.4, we regarded it as large. We set the criteria of the join factor [7] to 1. If $J_{p_i} > 1$, it is regarded as high, else low. The reason is that if the join factor is greater than 1, the join result size increases, else decreases. If we defer an expensive refinement operation in the join of decreasing size, there can be many chances to filter out the objects to be refined in the non-spatial join or the spatial join filter. However, in the join of increasing size, there can be many duplications of the objects to be refined in the join result. This may incur

Table 5
Parameters of II and SA

	Parameters	Values
II	Stopping condition	Equal time to SA
	Local minimum	$5 * n$ Consecutive uphill moves
SA	Initial temperature	$T_0 = 2 * \text{cost}(S_0)$
	Frozen	$T < 1$ and minS unchanged for the last 4 stages
	Equilibrium	$10 * n$
	Temperature reduction	$T_{\text{new}} = 0.80 * T_{\text{old}}$

[7] The join fortor is defined as $J = \frac{\text{the join result size}}{\text{the average of the join input sizes}}$. If the join factor is high (low), the join selectivity is also high (low).

the redundant I/O and computation. In this case, we evaluated all the unevaluated refinement operations before the join to reduce the join input size.

We analyzed the behavior of each algorithm for query types Q1, Q2 and Q3 along the variation of the selectivity for spatial and non-spatial operations and the complexity of spatial objects (the average number of points per spatial object) as follows:

(1) A1: the middle selectivity of selections and joins, and the low complexity of spatial objects ($S_{nsp} = S_{sp} = 1/4$, $J_{nsp} = J_{sp} = 0.8$, $pts = 20$),
(2) A2: the middle selectivity of selections and joins, and the high complexity of spatial objects ($S_{nsp} = S_{sp} = 1/4$, $J_{nsp} = J_{sp} = 0.8$, $pts = 100$),
(3) A3: the low selectivity of selections and the high selectivity of joins, and the high complexity of spatial objects ($S_{nsp} = S_{sp} = 1/16$, $J_{nsp} = J_{sp} = 4.8$, $pts = 100$),
(4) A4: the high selectivity of selections and the low selectivity of joins, and the high complexity of spatial objects ($S_{nsp} = S_{sp} = 1/2$, $J_{nsp} = J_{sp} = 0.1$, $pts = 100$).

4.1. Experimental results for 5-way select-join queries

The first experiment was conducted for the 5-way select-join query. Each query was performed five times for each selectivity and complexity. Fig. 3 shows the average scaled cost, which set the optimal cost of ESFAR to 1, found by the above six algorithms over time for the 5-way select-join of query type Q1 with the varying selectivity and complexity. We ran the II, SA, II_{is} and SA_{is} algorithms during 20 s. The optimization time for TRA is similar for A1, A2, A3 and A4. Likewise, that for ESFAR is also mutually similar. The average optimization time for TRA is about 4 s, and that for ESFAR is about 48 s. First, in the comparison of TRA and ESFAR, the effect of ESFAR in A2 is larger than that in A1. This means that the more complex spatial objects are, the larger the effect of ESFAR is. Among A2, A3 and A4, the ESFAR effect is best in A4, worst in A3. This means that as the selectivity of the selection is high and the selectivity of the join is low, the effect of ESFAR becomes large. The experimental results between TRA and ESFAR are similar to those in Section 2.5. However, the cost difference between TRA and ESFAR is higher for the complex queries than for the simple queries of Section 2.5.

Next, in the comparison of II and SA, the local minimum costs found by II decrease faster than those by SA in most cases, and the final costs of II in 20 s are lower than those of SA in many cases. However, in the case of A2, the final cost of SA is lower than that of II. II and SA with the `initial_state` heuristic, i.e., II_{is} and SA_{is}, start with much lower initial costs. As the case of II and SA, the cost of II_{is} decreases faster than that of SA_{is}. However, in the final cost, sometimes II_{is} is lower, sometimes SA_{is} is lower. In most cases, the heuristic algorithms for ESFAR find a much lower cost plan than the tradition optimal plan. In particular, II_{is} and SA_{is} are stabilized with the cost near to the ESFAR optimal plan within 4 s which is the average optimization time for finding the traditional optimal plan.

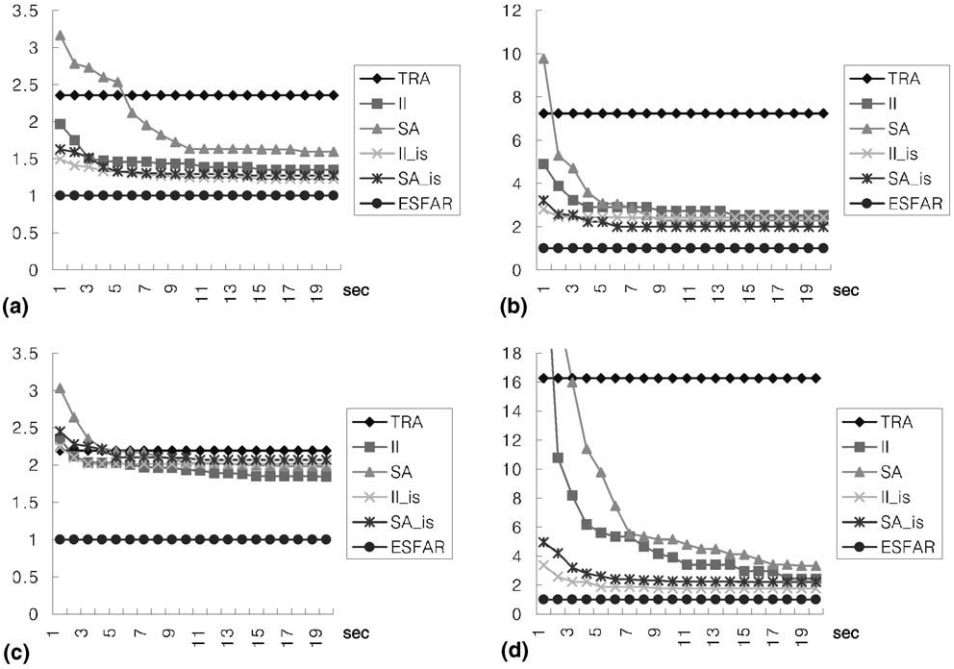

Fig. 3. Average scaled cost of plans over time for 5-way select-join queries: (a) A1; (b) A2; (c) A3; (d) A4.

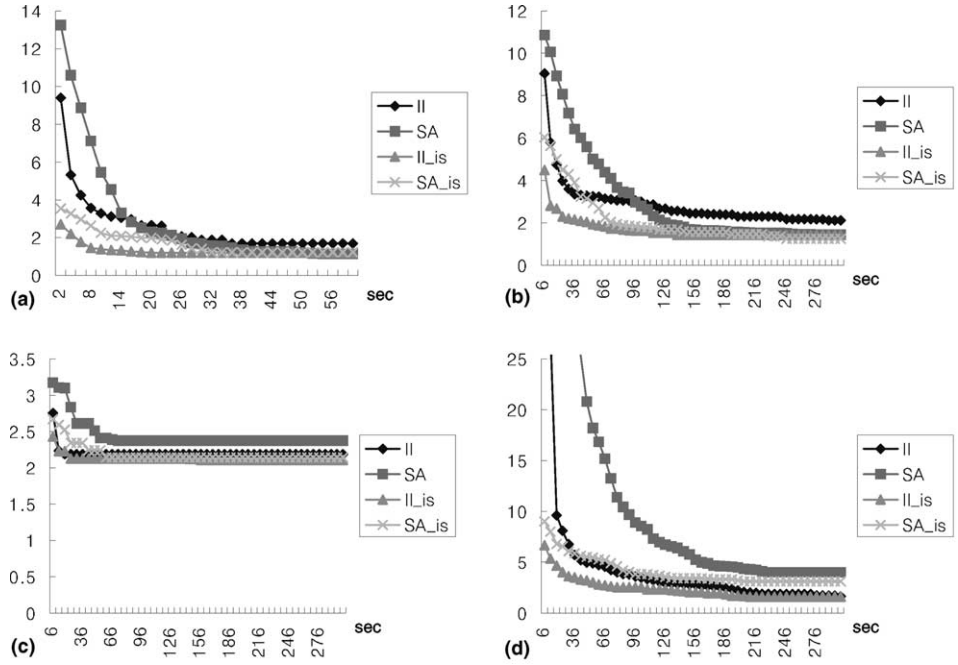

Fig. 4. Average scaled cost of plans over time for large select-join queries: (a) $n = 10$, A2; (b) $n = 20$, A2; (c) $n = 20$, A3; (d) $n = 20$, A4.

Fig. 5(a) compares the final costs after 20 s for the 5-way select-join of query types Q1, Q2 and Q3. In the cases of A2 and A4, for all query types, the costs of the ESFAR heuristic algorithms are much lower than the optimal plan cost of the traditional optimizer. However, in the cases of A1 and A3, the cost differences are small, and moreover in the case of A3 of Q3, the costs of the ESFAR heuristic algorithms are slightly higher than the optimal plan cost of the traditional optimizer. In the comparison between II and SA or between II_{is} and SA_{is}, while sometimes II (or II_{is}) is lower, sometimes SA (or SA_{is}) is lower. In many cases, the cost of II_{is} or SA_{is} is lower than that of II or SA. As shown in Figs. 3 and 5(a), in most cases, the plans produced by our heuristics perform better than those by TRA, and their costs have only small gab with the cost of the optimal ESFAR plan.

4.2. Experimental results for large select-join queries

Fig. 4 shows the scaled cost [8] of the heuristic algorithms over time for the large select-join of query type Q1 with the varying selectivity and object complexity. We cannot present the cost of TRA and ESFAR for the large select-join query because we cannot obtain the optimization result for the optimal algorithms due to the limit of the system resource. Figs. 4(a) and (b) compare the cost changes over time for the 10-way select-join and the 20-way select-join of query type Q1 when the slectivity and complexity are fixed to A2. The shapes of graphs over time are similar for both cases. In these cases, the final cost of SA is better than that of II in both cases. The initial costs found by the `initial_state` heuristic show a little difference. The initial cost for the 20-way select-join is higher than that for the 10-way select-join.

Fig. 4(b)–(d) compare the cost changes over time for the varying selectivity and complexity, A2, A3 and A4, in the 20-way select-join of query type Q1. The cost down over time is small for A3. A3 corresponds to the case that the ESFAR effect is small. The initial costs of all algorithms are higher for A4 than for A2 and the cost down is more rapid for A4 than for A2. A4 corresponds to the case that the ESFAR effect is large.

Fig. 5(b) shows the final costs of all heuristic algorithms for the large select-join of query types Q1, Q2 and Q3. In most cases, the final costs for II_{is} and/or SA_{is} are slightly better than those for II and/or SA. Summarizing, in most cases of Figs. 3–5, II_{is} and/or SA_{is} have lower initial costs than II and/or SA and rapidly converge to the final cost. Especially II_{is} converges faster than SA_{is} and has a similar final cost with SA_{is}.

5. Conclusions

Recently, there was a study about a query optimization technique which took the characteristics of spatial databases into account. The main idea of the study was to start the two-step processing of a spatial query, which

[8] The average scaled cost of a plan for the large select-join query is defined as the average cost of the plan divided by the lowest final cost of all plans for the specific query.

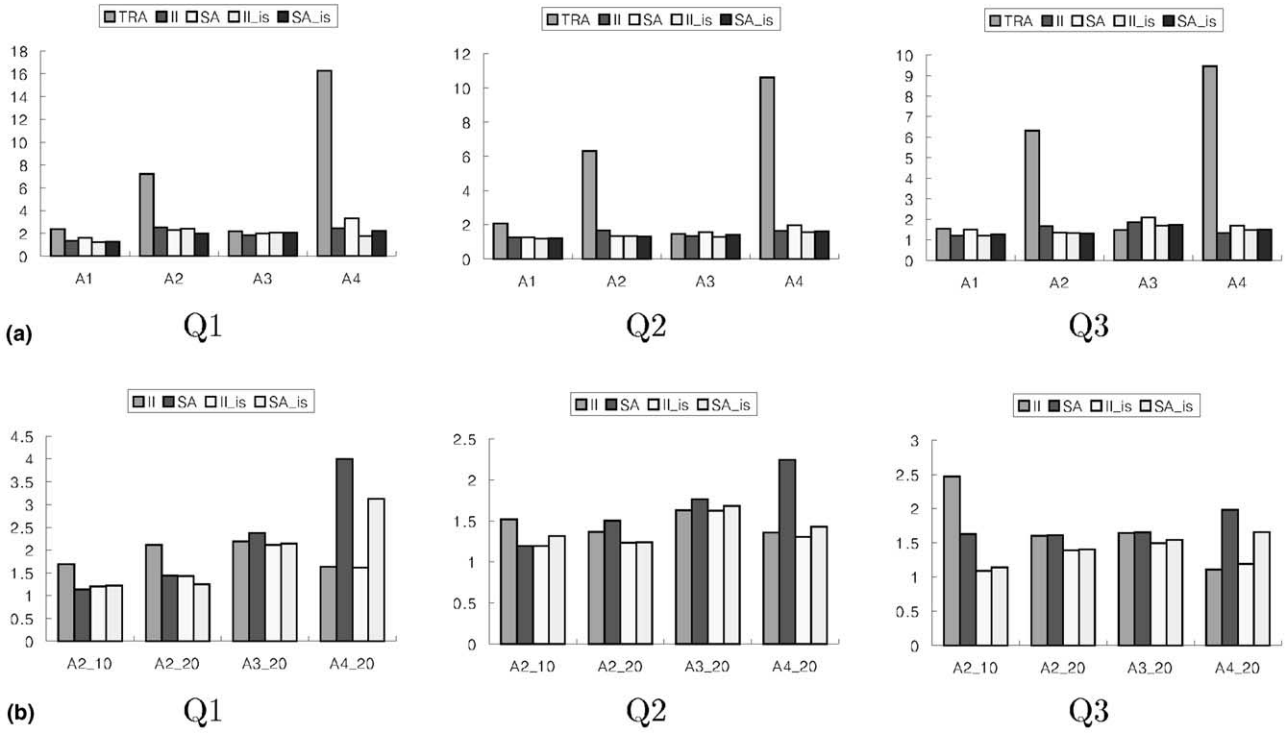

Fig. 5. Average scaled cost of the produced plans: (a) average scaled cost for 5-way select-join queries; (b) average scaled cost for large select-join queries.

has been applied only in the query execution phase, from the query optimization phase. The study 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 non-spatial operators or other spatial operators at the same level. This optimization technique was called ESFAR. In the study, an algebra, called ISOA, and optimization rules for ESFAR were also proposed.

The ESFAR optimization technique always generates more efficient execution plans than the traditional one because the ISOA operators include the traditional operators and the ESFAR rules include the traditional optimization rules. Because of more operators and more rules, the ESFAR optimizer consumes more optimization time than the traditional optimizer. In this paper, we applied two well-known heuristic search algorithms, II and SA, to the ESFAR optimizer. Additionally we developed a new heuristic algorithm to find a good initial state of II and SA.

In our previous study, we implemented the execution algorithms for ISOA. The experimental results using the TIGER data showed that the ESFAR optimization strategy generates more efficient query execution plans than the traditional one in many cases. In this paper, we improved the cost model of the ESFAR optimizer, and implemented the ESFAR optimizer, and ran the optimizer for the TIGER data, and compared the optimization results and the actual execution results. Especially our cost model considered the real life environment such as the LRU buffer, the clustering of the data, and the selectivity considering the real data distribution. The optimization results were similar with the actual execution results in most cases, and our optimizer estimated the lowest cost execution plan well. Wrong optimization results occurred only when the difference between the cheapest execution plan and the second cheapest plan was small.

VOG internally has the exhaustive search algorithm to generate the optimal plan. We added two heuristic search algorithms and our initial state heuristic to the implementation of our ESFAR optimizer. Through experiments, we showed that the heuristic algorithms for ESFAR find a good sub-optimal plan in a reasonable time. Mostly the heuristic algorithms found a lower cost plan than the optimal plan generated by the traditional optimizer. Especially the II algorithm with the initial state heuristic rapidly found a plan of a high quality.

Acknowledgements

We would like to thank Dr. Goetz Graefe and Dr. William J. McKenna for providing us with the source of the Volcano optimizer generator and the related documents. Volcano has been valuable for our research and development in query optimization. We also wish to

thank Dr. Robert Glass, editor-in-chief, for his help and the anonymous referees for their useful comments. This research was supported by the Basic Research Program (grant number 99-2-315-001-3) of the Korea Science and Engineering Foundation.

Appendix A. Cost model

In this section, we describe a cost model applied to the implementation of our optimizer. We assume that there is an LRU buffer in the system and all data are clustered on both a non-spatial attribute and a spatial attribute. 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.

A.1. Costs for non-spatial operations

We summarize cost models for non-spatial operations using references such as (Silberschatz et al., 1997; Ullman, 1988; Yao, 1977). Table 6 shows the statistics per class and attribute used for the computation of costs for non-spatial operations.

A.1.1. Non-spatial select

(1) N_Seq-Select

$$C_{\text{io}}(\text{N\_Seq\_Select}) = |R|. \tag{A.1}$$

(2) Btree-Select

$$\begin{aligned} C_{\text{io}}(\text{Btree\_Select}) &= \mathit{index\_page\_read} + \mathit{data\_page\_read} \\ &= \mathit{nonleaf\_page\_read} + \mathit{leaf\_page\_read} \\ &\quad + \mathit{data\_page\_read} \\ &= (h_R - 1) + \lceil k/f \rceil + Y(\|R\|, |R|, k), \\ \text{where } k &= Sp * \|R\|, \end{aligned} \tag{A.2}$$

Table 6
Statistics for non-spatial operations

Symbol	Meaning
$\|R\|, \|S\|$	Number of pages
$\|\|R\|\|, \|\|S\|\|$	Number of objects
$\|\|JR\|\|, \|\|JS\|\|$	Number of objects participating the join between R and S
N_l, M_l	Number of entries in level l of B+-tree in classes R and S
h_R, h_S	Height of B+-tree
f	Average fan-out of a B+-tree node
Sp, Jp	Selectivity for non-spatial select or non-spatial join
$V(R.A), V(S.A)$	Number of unique values for attribute A
$\min(R.A)$	Minimum value for attribute A
$\max(R.A)$	Maximum value for attribute A

$$Sp = \begin{cases} \frac{1}{V(R.A)} & \text{where } p = \text{`}=\text{'}, \\ \frac{\text{const} - \min(R.A)}{\max(R.A) - \min(R.A)} & \text{where } p = \text{`}<\text{'}, \text{`}\leqslant\text{'}, \\ \frac{\max(R.A) - \text{const}}{\max(R.A) - \min(R.A)} & \text{where } p = \text{`}>\text{'}, \text{`}\geqslant\text{'}, \end{cases} \tag{A.3}$$

$Y(n, m, k)$ in Eq. (A.2) stands for Yao's formula (Yao, 1977), which was approximated to the following equation in (Chung, 1983):

$$Y(n,m,k) = \begin{cases} k, & m = n, \\ 1, & m = 1, \\ m, & 1 < m < n,\ k > n - \frac{n}{m}, \\ m\left(1 - \left(1 - \frac{k}{n}\right)^{n/m}\right), & 1 < m < n,\ \frac{n}{m} < k \leqslant n - \frac{n}{m}, \\ m\left(1 - \left(1 - \frac{1}{m}\right)^{k}\right), & 1 < m < n,\ k < \min\left(n - \frac{n}{m}, \frac{n}{m}\right). \end{cases} \tag{A.4}$$

If all objects in class R are clustered on attribute A

$$C_{\text{cio}}(\text{Btree\_Select}) = (h_R - 1) + \lceil k/f \rceil + Sp * |R|. \tag{A.5}$$

A.1.2. Non-spatial join

(1) N_Indexed-Nested-Loop-Join

$$\begin{aligned} C_{\text{io}}(\text{N\_INLJ}) &= |R| + \|R\| * (\mathit{index\_page\_read} \\ &\quad + \mathit{data\_page\_read}) \\ &= |R| + \|R\| * (h_S + Jp * \|S\|), \\ \text{where } Jp &= \frac{1}{\max(V(R.A), V(S.A))}. \end{aligned} \tag{A.6}$$

If all objects in class S are clustered on attribute A

$$C_{\text{cio}}(\text{N\_INLJ}) = |R| + \|R\| * (h_S + Jp * |S|). \tag{A.7}$$

The indexed nested loop join performs the B-tree select operation repeatedly. If the system has an LRU buffer, the I/O cost will be greatly reduced. If the outer class is ordered along the indexed attribute of the inner class, the successive B-tree operations will access nearly the same nodes with a very small difference. Therefore, the miss ratio of the index access except the initial access will be close to zero. The I/O cost for the clustered and buffered indexed nested loop join is the following equation:

$$C_{\text{cbio}}(\text{N\_INLJ}) = |R| + J'_p * \left(\sum_{l=0}^{h_S - 1} N_l + |S| \right), \tag{A.8}$$

where

$$N_l = \frac{\|S\|}{f^l},$$

$$J'_p = \frac{\min(\max(R.A), \max(S.A))}{\max(S.A) - \min(S.A)} - \frac{\max(\min(R.A), \min(S.A))}{\max(S.A) - \min(S.A)}.$$

(2) Btree-Join

$$C_{\text{io}}(\text{Btree\_Join}) = \textit{index\_page\_read} + \textit{data\_page\_read} = h_R - 1 + h_S - 1 + \lceil \|R\|/f \rceil + \lceil \|S\|/f \rceil + \|JR\| + \|JS\|, \quad \text{(A.9)}$$

where

$$J = \frac{\|R\| * V(S.A)}{\max(V(R.A), V(S.A))},$$

$$JS = \frac{\|S\| * V(R.A)}{\max(V(R.A), V(S.A))}.$$

If all objects in classes R and S are clustered on attribute A

$$C_{\text{cio}}(\text{Btree\_Join}) = h_R - 1 + h_S - 1 + \lceil \|R\|/f \rceil + \lceil \|S\|/f \rceil + |R| + |S|. \quad \text{(A.10)}$$

(3) Non-spatial hash-join

The I/O cost for the non-spatial hash join (NHJ) was presented in Silberschatz et al. (1997). The hash join consists of the partition phase and the join phase. The I/O cost formulae for NHJ are like the following:

$$C_{\text{part}} = 2 * (|R| + |S|), \quad \text{(A.11)}$$

$$C_{\text{join}} = |R| + |S|, \quad \text{(A.12)}$$

$$C_{\text{io}}(\text{NHJ}) = C_{\text{part}} + C_{\text{join}} = 3 * (|R| + |S|). \quad \text{(A.13)}$$

A.2. Costs for spatial operations

We summarize the following equations with references to the cost models for spatial operations in Huang et al. (1997), Kamel and Faloutsos (1993), Lee and Chung (2000), Pagel et al. (1993), Theodoridis and Sellis (1996), Theodoridis et al. (1998). We assume that all spatial objects are normalized to fit within the unit space $[0,1]^2$. The statistics per class and attribute used for the computation of costs for spatial operations are shown in Table 7 in addition to Table 6.

A.2.1. Spatial select

(1) S_Seq-Select

$$C_{\text{io}}(\text{S\_Seq\_Select}) = |R|, \quad \text{(A.14)}$$

$$C_{\text{cpu}}(\text{S\_Seq\_Select}) = \|R\| * T_{\text{gc}}. \quad \text{(A.15)}$$

In Eq. (A.15), T_{gc} means the geometric computation time for a spatial object and is represented by Eq. (A.16) for the spatial select operation such as the window query

$$T_{\text{gc}} = v * T_{\text{edge-rect}}. \quad \text{(A.16)}$$

(2) Rtree-Select

Table 7
Statistics for spatial operations

Symbol	Meaning
N, M	Total number of MBR for spatial objects in classes R and S
N_l, M_l	Number of MBR in level l of R*-tree in classes R and S
$h(h_R, h_S)$	Height of R*-tree in classes R and S
f	Average fan-out of an R*-tree node
n_{ix}, m_{ix}, n_{iy}, m_{iy}	x-Length and y-length of MBR for a spatial object i in classes R and S
n_{lix}, m_{lix}, n_{liy}, m_{liy}	x-Length and y-length of MBR i in level l of R*-tree in classes R and S
n_x, m_x, n_y, m_y	Average x-length and y-length of MBRs for spatial objects in classes R and S
$n_{lx}, m_{lx}, n_{ly}, m_{ly}$	Average x-length and y-length of MBRs in level l of R*-tree in classes R and S
v, w	Average number of points for spatial objects in classes R and S
N^*	Number of window queries until the steady state buffer hit probability

$$C_{\text{io}}(\text{Rtree\_Select}) = \textit{index\_page\_read} + \textit{data\_page\_read} = \sum_{l=0}^{h-1} \sum_{i=1}^{N_l} (n_{lix} + q_x) * (n_{liy} + q_y) + Y(\|R\|, |R|, k) = \sum_{l=0}^{h-1} N_l * (n_{lx} + q_x) * (n_{ly} + q_y) + Y(\|R\|, |R|, k), \quad \text{(A.17)}$$

where

$$k = \sum_{i=1}^{N} (n_{ix} + q_x) * (n_{iy} + q_y) = N * (n_x + q_x) * (n_y + q_y).$$

If all objects in class R are clustered on the spatial attribute

$$C_{\text{cio}}(\text{Rtree\_Select}) = \sum_{l=0}^{h-1} N_l * (n_{lx} + q_x) * (n_{ly} + q_y) + |R| * (n_x + q_x) * (n_y + q_y), \quad \text{(A.18)}$$

$$C_{\text{cpu}}(\text{Rtree\_Select}) = \textit{number\_of\_candidates} * \textit{geometric\_computation\_time} = N_{\text{cand}} * T_{\text{gc}}. \quad \text{(A.19)}$$

Calculating candidate objects in a window query is presented in Lee and Chung (2000) and represented by Eqs. (A.20) and (A.21). N_{hit} in Eq. (A.20) is for objects that were already hit in the filter step and do not need the geometric computation, and (N_{cand} in Eq. (A.21) is for candidate objects that need the geometric computation.

$$N_{\text{hit}} = \begin{cases} N * \big((n_x + q_x) * (n_y + q_y) - 4 * n_x * n_y\big), \\ \quad q_x > n_x,\ q_y > n_y, \\ 0, \quad \text{otherwise}, \end{cases} \tag{A.20}$$

$$N_{\text{cand}} = 4 * n_x * n_y. \tag{A.21}$$

Theodoridis and Sellis (1996) derived the non-leaf level parameters of R-tree using only the amount and density of spatial objects. The density of spatial objects can be represented by an average MBR size

$$h = 1 + \left\lceil \log_f \frac{N}{f} \right\rceil, \tag{A.22}$$

$$N_l = \frac{N}{f^l}, \tag{A.23}$$

$$n_{l+1,x} = \left(\sqrt{f} - 1\right) * \frac{1}{\sqrt{N_l}} + n_{l,x}, \tag{A.24}$$

$$n_{l,x} = \frac{1}{\sqrt{N}} \left(\sqrt{f}^{\,l} - \sqrt{f}\right) + n_x. \tag{A.25}$$

A.2.2. Spatial join

(1) S_Indexed-Nested-Loop-Join

$$\begin{aligned} &C_{\text{io}}(\text{S\_INLJ}) \\ &\quad = |R| + \|R\| * (\mathit{index\_page\_read} + \mathit{data\_page\_read}) \\ &\quad = |R| + \|R\| * \left(\sum_{l=0}^{h_S - 1} M_l * (m_{lx} + n_x) * (m_{ly} + n_y) \right. \\ &\qquad \left. + M * (m_x + n_x) * (m_y + n_y) \right). \end{aligned} \tag{A.26}$$

If all objects in class S are clustered on the spatial attribute

$$\begin{aligned} &C_{\text{cio}}(\text{S\_INLJ}) \\ &\quad = |R| + \|R\| * \left(\sum_{l=0}^{h_S - 1} M_l * (m_{lx} + n_x) * (m_{ly} + n_y) \right. \\ &\qquad \left. + |S| * (m_x + n_x) * (m_y + n_y) \right), \end{aligned} \tag{A.27}$$

$$\begin{aligned} C_{\text{cpu}}(\text{S\_INLJ}) &= \|R\| * \mathit{number\_of\_candidates} \\ &\quad * \mathit{geometric\_computation\_time} \\ &= \|R\| * M * (m_x + n_x) * (m_y + n_y) * T_{\text{gc}}. \end{aligned} \tag{A.28}$$

In Eq. (A.28), T_{gc} is the geometric computation time for the spatial join. We assume only the intersection join. The intersection join can be implemented by the plane sweep algorithm. The computation time for the plane sweep algorithm is represented by Eq. (A.29)

$$T_{\text{gc}} = (v + w) \log_2(v + w) * T_{\text{edge-edge}}. \tag{A.29}$$

The indexed nested loop join performs the repeated window queries. If the system has an LRU buffer, the actual I/O time will be much smaller than Eq. (A.26). In an LRU buffered environment, a cost model for the window query was presented in Leutenegger and Lopez (1998). According to Leutenegger and Lopez (1998), Eq. (A.18) must be changed as the following equation in an LRU buffered environment:

$$\begin{aligned} C_{\text{cbio}}(\text{Rtree\_Select}) &= \sum_{l=0}^{h-1} N_l * (n_{lx} + q_x) * (n_{ly} + q_y) \\ &\quad * \big(1 - (n_{lx} + q_x) * (n_{ly} + q_y)\big)^{N^*} \\ &\quad + |R| * (n_x + q_x) * (n_y + q_y). \end{aligned} \tag{A.30}$$

Therefore, the I/O cost for the indexed nested loop join in a clustered and buffered environment is like the following:

$$\begin{aligned} C_{\text{cbio}}(\text{S\_INLJ}) &= |R| + \|R\| * \left(\sum_{l=0}^{h_S - 1} M_l * (m_{lx} + n_x) \right. \\ &\quad * (m_{ly} + n_y) * (1 - (m_{lx} + n_x) \\ &\quad \left. * (m_{ly} + n_y))^{N^*} \right) + |S| * Q_x * Q_y, \end{aligned} \tag{A.31}$$

where

$Q_x * Q_y = \textit{an MBR enclosing all rectangles in class } R.$

(2) Rtree-Join

The I/O time for the R-tree join was presented in Huang et al. (1997), Theodoridis et al. (1998). Especially in Huang et al. (1997), the model estimating the number of I/O for the R-tree join when LRU buffers exist was presented. The I/O time for the R-tree join in the case of no buffering is represented by Eq. (A.32) and the CPU time for the refinement step is represented by Eq. (A.34)

$$\begin{aligned} &C_{\text{io}}(\text{Rtree\_Join}) \\ &\quad = \mathit{index\_page\_read} + \mathit{data\_page\_read} \\ &\quad = \sum_{l=0}^{h-1} \sum_{i=1}^{N_l} \sum_{j=1}^{M_l} (n_{lix} + m_{ljx}) * (n_{liy} + m_{ljy}) * 2 \\ &\qquad + \sum_{i=1}^{N} \sum_{j=1}^{M} (n_{ix} + m_{jx}) * (n_{iy} + m_{jy}) * 2 \\ &\quad = \sum_{l=0}^{h-1} N_l * M_l * (n_{lx} + m_{lx}) * (n_{ly} + m_{ly}) * 2 \\ &\qquad + N * M * (n_x + m_x) * (n_y + m_y) * 2. \end{aligned} \tag{A.32}$$

If all objects in classes R and S are clustered on the spatial attribute

$$\begin{aligned} C_{\text{cio}}(\text{Rtree\_Join}) &= \sum_{l=0}^{h-1} N_l * M_l * (n_{lx} + m_{lx}) \\ &\quad * (n_{ly} + m_{ly}) * 2 + |R| + |S|, \end{aligned} \tag{A.33}$$

$$C_{\text{cpu}}(\text{Rtree\_Join}) = \textit{number\_of\_candidates} * \textit{geometric\_computation\_timeN} * M * (n_x + m_x) * (n_y + m_y) * T_{\text{gc}}. \quad (A.34)$$

As in the case of Huang et al. (1997), if we denote the *index_page_read* portion of $C_{\text{io}}(\text{Rtree\_Join})$ as ZEIO (zero-buffer expected I/O),

$$\text{ZEIO} = \sum_{l=0}^{h-1} N_l * M_l * (n_{lx} + m_{lx}) * (n_{ly} + m_{ly}) * 2. \quad (A.35)$$

According to Huang et al. (1997), the expected I/O (EIO) of the R-tree join in an LRU buffered environment is estimated by the following equation:

$$\text{EIO} = P + Q + (\text{ZEIO} - P - Q) * P\{x \geqslant b\}, \quad \text{where } P = \sum_{l=1}^{h} N_l,\ Q = \sum_{l=1}^{h} M_l. \quad (A.36)$$

In the above equation, x denotes a random variable which represents the number of page faults between two consecutive accesses for an R*-tree node, and b denotes the number of buffer entries in a system. Huang et al. (1997) applied the exponential distribution as the distribution function of x. Therefore, $P\{x \geqslant b\}$ in the above equation is calculated like the following:

$$f(x) = \lambda e^{-\lambda x}, \quad \text{where } \lambda = \frac{1}{\mu}, \quad (A.37)$$

$$F(x) = 1 - e^{-\lambda x}, \quad (A.38)$$

$$P\{x \geqslant b\} = 1 - F(b). \quad (A.39)$$

The value of μ, i.e., the mean of x, was approximated to the following (Huang et al., 1997):

$$\mu = c * \frac{\text{ZEIO}}{P + Q}. \quad (A.40)$$

Consequently, the I/O cost for the R-tree join in a clustered and buffered environment is estimated as follows:

$$C_{\text{cbio}}(\text{Rtree\_Join}) = \text{EIO} + |R| + |S|. \quad (A.41)$$

(3) Spatial hash-join

The I/O cost for the SHJ was presented in Koudas and Sevcik (1997), Mamoulis and Papadias (2000). Generally the hash join consists of the partition phase and the join phase. For partitioning of datasets, both datasets are read and hashed into buckets. The partitioning cost of SHJ is given by the following formula: r_S and f_S represent the replication and filtering ratios of $|S|$:

$$C_{\text{part}} = 2 * |R| + (2 + r_S - f_S) * |S|. \quad (A.42)$$

Next, the data from each bucket pair will be joined. The join cost of SHJ is

$$C_{\text{join}} = |R| + (1 + r_S - f_S) * |S|. \quad (A.43)$$

Summarizing, the total I/O cost of SHJ considering an LRU buffer is

$$C_{\text{io}}(\text{SHJ}) = C_{\text{part}} + C_{\text{join}} = 3 * |R| + (3 + r_S - f_S) * |S|. \quad (A.44)$$

The CPU time for SHJ is the same as that for the R-tree join, i.e., Eq. (A.34).

References

Acharya, S., Poosala, V., Ramaswamy, S., 1999. Selectivity estimation in spatial databases. Proc. ACM SIGMOD, 487–498.

Aref, W., Samet, H., 1991. Optimization strategies for spatial query processing. Proc. VLDB, 81–90.

Becker, L., Güting, R.H., 1992. Rule-based optimization and query processing in an extensible geometric database system. ACM Trans. Database Syst. 17 (2), 247–303.

Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B., 1990. The R*-tree: an efficient and robust access method for points and rectangles. Proc. ACM SIGMOD, 322–331.

Blasgen, M., Eswaran, K., 1977. Storage and access in relational databases. IBM Syst. J. 16 (4), 363–377.

Brinkhoff, T., Kriegel, H.-P., Seeger, B., 1993. Efficient processing of spatial joins using R-trees. Proc. ACM SIGMOD, 237–246.

Cho, W.-S., Whang, K.-Y., Lee, S.-S., Yoon, Y.-I., 1997. Query optimization techniques utilizing path indexes in object-oriented database systems. Proc. DASFAA, 21–29.

Chung, C.-W., 1983. A query optimization in distributed database systems. Ph.D. Dissertation, University of Michigan.

Graefe, G., McKenna, W.J., 1993. The volcano optimizer generator: extensibility and efficient search. Proc. IEEE ICDE, 209–218.

Huang, Y.-W., Jing, N., Rundensteiner, E.A., 1997. A cost model for estimating the performance of spatial joins using R-trees. In: Proceedings of the Ninth International Conference on Scientific and Statistical Database Management, pp. 30–38.

Ioannidis, Y.E., Kang, Y.C., 1990. Randomized algorithms for optimizing large join queries. Proc. ACM SIGMOD, 312–321.

Kabra, N., DeWitt, D., 1996. Opt++: an object-oriented implementation for extensible database query optimization. VLDB J. 8 (1), 55–78.

Kamel, I., Faloutsos, C., 1993. On packing R-tree. Proc. CIKM, 490–499.

Koudas, N., Sevcik, K.C., 1997. Size separation spatial join. Proc. ACM SIGMOD, 324–355.

Lee, Y.-J., Chung, C.-W., 2000. Analysis of two-step index structure for complex spatial objects. Inf. Sci. 125 (1–4), 133–152.

Leutenegger, S.T., Lopez, M.A., 1998. The effect of buffering on the performance of R-trees. Proc. IEEE ICDE, 164–171.

Mamoulis, N., Papadias, D., 2000. Multiway spatial joins. Technical Report HKUST-CS00-01, Hong Kong University of Science and Technology.

Mohan, C., Haderlr, D., Wang, Y., Cheng, J., 1990. Single table access using multiple indexes: optimization, execution, and concurrency control techniques. Proc. EDBT, 29–43.

Ooi, B.C., Sacks-Davis, R., McDonell, K.J., 1989. Extending a DBMS for geographic applications. Proc. IEEE ICDE, 590–597.

Orenstein, J.A., 1986. Spatial query processing in an object-oriented database system. Proc. ACM SIGMOD, 326–336.

Pagel, B., Six, H., Toben, H., Widmayer, P., 1993. Towards an analysis of range query performance in spatial data structures. Proc. ACM PODS, 214–221.

Papadias, D., Theodoridis, Y., Sellis, T., Egenhofer, M.J., 1995. Topological relations in the world of minimum bounding rectangles: a study with R-trees. Proc. ACM SIGMOD, 92–103.

Park, H.-H., Hong, N.-H., Park, C.-W., Chung, C.-W., 1997. Spatial object algebra and query language in OMEGA. Technical Report CS/TR-97-118, Korea Advanced Institute of Science and Technology.

Park, H.-H., Lee, C.-G., Lee, Y.-J., Chung, C.-W., 1999. Early separation of filter and refinement steps in spatial query optimization. Proc. DASFAA, 161–168.

Park, H.-H., Lee, Y.-J., Chung, C.-W., 2000. Spatial query optimization utilizing early separated filter and refinement strategy. Inf. Syst. 25 (1), 1–22.

Park, H.-H., 2001. Early separated filter and refinement strategy and multi-way spatial joins for spatial query optimization. Ph.D. Thesis, Korea Advanced Institute of Science and Technology.

Patel, J. et al., 1997. Building a scalable geo-spatial DBMS: technology, implementation, and evaluation. Proc. ACM SIGMOD, 336–347.

Piatetsky-Shapiro, G., Connell, C., 1984. Accurate estimation of the number of tuples satisfying a condition. Proc. ACM SIGMOD, 256–276.

Samet, H., Aref, W.G., 1995. Spatial data models and query processing. In: Modern Database Systems. ACM Press, New York, pp. 338–360.

Selinger, P.G. et al., 1979. Access path selection in a Relational Database Management System. Proc. ACM SIGMOD, 23–34.

Silberschatz, A., Korth, H.F., Sudarshan, S., 1997. Database System Concepts, third ed. McGraw-Hill, New York.

Theodoridis, Y., Sellis, T., 1996. A model for the prediction of R-tree performance. Proc. ACM PODS, 161–171.

Theodoridis, Y., Stefanakis, E., Sellis, T., 1998. Cost models for join queries in spatial databases. Proc. IEEE ICDE, 476–483.

US Bureau of the Census, 1995. TIGER/Line Files. Technical Documentation.

Ullman, J.D., 1988. In: Principles of Database and Knowledge-Base Systems, vols. 1 and 2. Computer Science Press, Rockville, MD.

Valduriez, P., 1987. Join indices. ACM Trans. Database Syst. 12 (2), 218–246.

Yao, S.B., 1977. Approximating the number of accesses in database organizations. Commun. ACM 20 (4), 260–261.

Ho-Hyun Park received his BS degree in the Department of Computer Science and Statistics, Seoul National University in Korea in 1987 and MS and Ph.D. degree in the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST) in 1995 and 2001. He currently works for Samsung Electronics. His research interests include geographic information systems, spatial databases, query processing, and home/office networks.

Hyung-Ju Cho received his BS and MS degree in the Department of Computer Engineering, Seoul National University in Korea in 1997 and 1999. He is currently a PhD student in the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST). His research interests include indexing and query processing techniques and query optimization in spatial, temporal, and spatio-temporal databases.

Chin-Wan Chung is a professor in the Department of Computer Science at the Korea Advanced Institute of Science and Technology (KAIST) since 1993. Prior to that, he was a senior research scientist and a staff research scientist in the Computer Science Department at the General Motors Research Laboratories (GMR). He received a Ph.D. from the University of Michigan in 1983. While at GMR, he developed DATAPLEX, a heterogenous distributed database management system integrating relational databases and hierarchical databases. At KAIST, he developed a full scale object-oriented spatial database management system called OMEGA, which supports ODMG standards. His current research interests include spatio-temporal databases, XML, multimedia databases, and Web databases.