

A Cost Model for Spatial Intersection Queries on RI-Trees

Hans-Peter Kriegel^{*}, Martin Pfeifle^{*}, Marco Pötke^{**} and Thomas Seidl^{***}
^{*}University of Munich, {kriegel, pfeifle}@dbs.informatik.uni-muenchen.de
^{**}sd&m AG software design & management, marco.poetke@sdm.de
^{***}RWTH Aachen University, seidl@informatik.rwth-aachen.de

Abstract. The efficient management of interval sequences represents a core requirement for many temporal and spatial database applications. With the Relational Interval Tree (RI-tree), an efficient access method has been proposed to process intersection queries of spatial objects encoded by interval sequences on top of existing object-relational database systems. This paper complements that approach by effective and efficient models to estimate the selectivity and the I/O cost of interval sequence intersection queries in order to guide the cost-based optimizer whether and how to include the RI-tree into the execution plan. By design, the models immediately fit to common extensible indexing/optimization frameworks, and their implementations exploit the built-in statistics facilities of the database server. According to our experimental evaluation on an Oracle database, the average relative error of the estimated query results and costs lies in the range of 0% to 32%, depending on the size and the structural complexity of the query objects.

1 Introduction

After two decades of temporal and spatial index research, the efficient management of one- and multi-dimensional extended objects has become an enabling technology for many novel database applications. The interval, or, more generally, the sequence of intervals, are basic datatypes for temporal and spatial data.

Highly accurate but still efficient selectivity estimation and cost prediction are the fundamentals of effective query optimization. For complex query objects and query predicates, the recent object-relational database servers provide extensible optimization frameworks that come along with the extensible indexing frameworks, in order to complete the seamless integration of user-defined index structures and appropriate cost models into the declarative DML. As an example for such an extension, we propose a cost model for interval sequence intersection queries on the spatial variant of the RI-tree [4]. This cost model is an extension of the cost model presented in [2] which is tailor-made for interval intersection queries on the basic RI-tree [3]. To get a suitable starting point for this paper, we suggest to have a look into the three above mentioned papers or into the technical report [1] belonging to this paper.

The remainder of this paper is organized as follows. In Section 2 we propose an approach to estimate the selectivity and the I/O cost of intersection queries on interval sequences. After an empirical evaluation of the presented methods in Section 3, the paper is concluded in Section 4.

2 Cost Estimation for Interval Intersection Queries

The functions presented in [2] for selectivity and cost estimation of interval intersection queries on the basic variant of the RI-tree [3], can naturally be extended to interval sequences. To enable query optimization for spatial queries on multidimensional extended objects, the corresponding interval sequence query for the chosen space-filling curve has to be evaluated. Unfortunately, a straightforward application of the proposed techniques to each single interval of the interval sequence can be too inefficient due to the following reasons:

- A spatial query is specified by an extended, multidimensional object. For arbitrary query objects, in particular objects not already indexed in the RI-tree, we have to compute the corresponding interval sequence before the above functions can be applied. This would typically be the case for window or box queries.
- As the cardinality of the interval sequence of a spatial object is proportional to its multidimensional surface the estimation for each single interval consumes very much CPU cost in the context of pure query optimization, even if the interval sequence of the spatial object has already been computed.

Thus, the evaluation on the full interval sequence anticipates substantial parts of the merely potential query processing on the RI-tree. The extensible optimizer might choose another access path instead, e.g. a full table scan on the base table or an index scan for a non-spatial predicate. In this case, the interval sequence filter may be skipped, and the query processor may refine the spatial predicate directly on the accurate representation of the spatial objects, e.g. on the polygons of a GIS database. Thus, the spatial decomposition and linearization of the query object into intervals would have been done in vain. The following paragraphs address these issues by extending the selectivity estimation and the cost model of [2] to a coarse aggregation of the potential query interval sequence. For processing interval sequence intersections, we will rely on the optimized approach of the RI-tree as presented in [4].

2.1 Aggregates on Interval Sequences

In order to minimize the cost of generating and evaluating the fine-grained interval sequence F of a spatial query object, we compute a coarse size-bound approximation C which conservatively approximates F with c intervals. The bound c could be set to a low constant number of intervals, or alternatively, c may depend on the spatial extension of the query object.

Let $F = (\psi_1, \psi_2, \dots, \psi_f)$ be the fine-grained interval sequence which would be the result of a granularity-, size- or error-bound decomposition of the spatial query object. Let $C = (\phi_1, \phi_2, \dots, \phi_c)$ be a coarse conservative approximation of the query interval sequence, where $c \ll f$. The basic idea is to materialize and use C instead of F for the query optimization phase, because deriving an estimation of the selectivity and the cost from F would already reach the CPU complexity of processing the exact query itself.

We first define an intersection ranking function $\rho_{intersect}$ and, based on this definition, two aggregates *coverage* and *cardinality* on interval sequences.

Definition 1 (Intersection Ranking Function)

Let $B \subseteq \mathfrak{R}$ be a domain of interval bounds and let $D = \{(l, u) \in B^2 \mid l \leq u\}$ be the corresponding interval domain. For intervals $\tau = (l_\tau, u_\tau) \in D$ and $\kappa = (l_\kappa, u_\kappa) \in D$, the *intersection ranking function*, $\rho_{\text{intersect}}: D \times D \rightarrow \mathfrak{R}$, is defined by

$$\rho_{\text{intersect}}(\tau, \kappa) = \begin{cases} \min(u_\tau, u_\kappa) - \max(l_\tau, l_\kappa), & \text{if } \tau \text{ and } \kappa \text{ intersect;} \\ 0, & \text{otherwise.} \end{cases}$$

Definition 2 (Aggregates on Interval Sequences)

Let $F = (\psi_1, \psi_2, \dots, \psi_f)$, $f \geq 1$, and $C = (\phi_1, \phi_2, \dots, \phi_c)$, $c \geq 1$, be two interval sequences representing the same spatial object. Let C be a conservative approximation of F , i.e. $c \leq f$ and

$$\bigcup_{i=1 \dots f} \psi_i \subseteq \bigcup_{i=1 \dots c} \phi_i.$$

For each i , $1 \leq i \leq c$, we define the following *aggregates* on F with respect to C , where $\rho_{\text{intersect}}$ is the intersection ranking according to Definition 1:

- (i) The *coverage* along ϕ_i of F is $\text{coverage}(\phi_i, F) = \sum_{j=1}^f \rho_{\text{intersect}}(\phi_i, \psi_j)$.
- (ii) The *cardinality* along ϕ_i of F is $\text{cardinality}(\phi_i, F) = |\{j, 1 \leq j \leq f \mid \phi_i \text{ intersects } \psi_j\}|$.

Thus, *coverage* denotes the fine-grained hyper-volume within a single interval of the coarse interval sequence C , while *cardinality* denotes the number of the respective fine-grained intervals.

2.2 Extended Selectivity Estimation

For each interval $\phi_i = (l_i, u_i)$ in C , we can assume a data space of $\text{coverage}(\phi_i, F)$ to be occupied by the corresponding intervals in F . Starting from a selectivity estimation $\sigma(I, \phi_i)$ for an intersection query ϕ_i on the intervals I [2], we estimate the total selectivity $\sigma(I, C, F)$ for the fine-grained interval sequence F by

$$\sigma(I, C, F) = \sum_{i=1}^c \left(\sigma_i(I, \phi_i) \cdot \frac{\text{coverage}(\phi_i, F)}{u_i - l_i} \right),$$

where $\sigma_1(I, \phi_1) = \sigma(I, \phi_1)$. The computation of $\sigma_i(I, \phi_i)$, $1 < i \leq c$, is similar to $\sigma(I, \phi_i)$, but considers only the portions of the ranges for r_{left} , r_{inner} and r_{right} which have not already contributed to $\sigma_{i-1}(I, \phi_{i-1})$. Thereby, the range queries on the quantile statistics [2] are kept disjoint, and $0 \leq \sigma(I, C, F) \leq 1$ holds.

2.3 Extended I/O Cost Model

Next, we present a I/O cost model for interval intersection queries on the spatial variant of the RI-tree. The computation of $\text{output}_{I/O}$ for a fine-grained interval sequence F can be done straightforward as explained in [2], based on an appropriate selectivity estimation. On the other hand, the non-blocked overhead $\text{join}_{I/O}$ has to be estimated for a sequence of interval intersections. Our model relies on the following two observations:

- In a real user environment with many concurrent queries, substantial parts of the B^+ -directories typically reside in the main memory and can be managed by the built-in LRU-cache of the DBMS [5]. Thus, it is not too farfetched, if we assume that

the complete B^+ directory is in the cache, especially if we consider interval sequences.

- The transient join partners are processed in increasing order (*left queries*, *inner query*) or decreasing order (*right queries*) with respect to the *node* value in the composite indexes on (*node*, *upper*, *id*) and (*node*, *lower*, *id*), respectively. Due to this ordered access, pages that are read several times during query processing will rarely be displaced from the LRU cache between the accesses. We therefore assume that each leaf page is retrieved only once from secondary storage.

In particular, the locality assumption of random I/Os is correct, as for sorted query sequences, the transient join partners are generated in ascending order over all gaps [4]. The local descending ordering of the *right queries* within each gap is typically absorbed even by small LRU caches. Thus, the approach presented in [2] to estimate $join_{I/O}$ is applicable with the slight modification that the fine-grained intervals of F are not known this time.

For each interval $\varphi_i = (l_i, u_i)$ in C , we assume a number of $cardinality(\varphi_i, F)$ of covered intervals in F . The average interval length among these intervals is

$$avgInt(\varphi_i, F) = \frac{coverage(\varphi_i, F)}{cardinality(\varphi_i, F)}.$$

Similarly, the average gap length along φ_i is estimated by

$$avgGap(\varphi_i, F) = \frac{(u_i - l_i) - coverage(\varphi_i, F)}{cardinality(\varphi_i, F) - 1}.$$

For each φ_i , the number and distribution of contained *left* and *right* queries could be estimated, and the multi-sets $NGaps_{left}(\varphi_i, F)$ and $NGaps_{right}(\varphi_i, F)$ of query gaps among the fine-grained representation of φ_i in F could be populated similarly as in [2]. But, as this extensive analysis would already be based on the estimated input of C and would increase the computational complexity of the cost model, we simply assume that the mean value of $NGaps_{right}(\varphi_i, F)$ is equal to the sum of $avgGap(\varphi_i, F)$ and $avgInt(\varphi_i, F)$. As the *inner queries* are integrated with the *left queries*, the mean value of $NGaps_{left}(\varphi_i, F)$ then corresponds to $avgGap(\varphi_i, F)$. Thereby, we subsume all potential queries within gaps of F by single *left* and single *right queries* (cf. Figure 1):

- (i) $avg(NGaps_{left}(\varphi_i, F)) = avgGap(\varphi_i, F),$
- (ii) $avg(NGaps_{right}(\varphi_i, F)) = avgGap(\varphi_i, F) + avgInt(\varphi_i, F).$

The respective averages on the block gaps $BGaps_{right}(\varphi_i, F)$ and $BGaps_{left}(\varphi_i, F)$ can be computed as presented in [2].

When applied to the lexicographic ordering, these plain averages suffice since particularly for convex spatial objects, the variance among the interval gaps is very small. Unfortunately, on the far more powerful concept of fractal space filling curves, including the Z- and Hilbert-ordering, the variance among the gap lengths is extremely high. In this case, the above averages provide a very weak characterization for the expected gap distribution. We have observed that the binary logarithms of fractal gaps typically obey an exponential distribution. Furthermore, histograms on fractal gaps show local

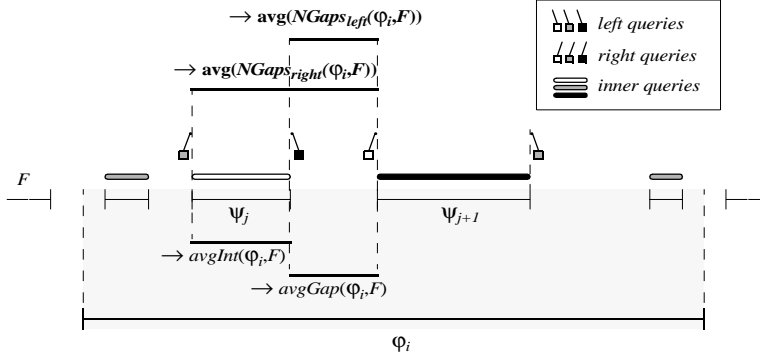


Figure 1: Contributors (\rightarrow) to the averages $avgInt(\phi_i, F)$ and $avgGap(\phi_i, F)$ for a coarse interval, and the mean of $NGaps_{left}(\phi_i, F)$ and $NGaps_{right}(\phi_i, F)$.

peaks at whole multiples of the original data dimension d , i.e. at gap lengths around 2^{k-d} , $k \geq 0$. This behavior is caused by the fact that many gaps represent empty square-shaped (2D) or cubical (3D) regions at the boundary of spatial objects. Our proposed estimation gap_{est} of this distribution is as simple as effective: the overall exponential shape is approximated by a geometric distribution on the base points $p_k = (2^{k-d})_k$, $k \geq 0$. We use a constant mean value μ according to the complexity of the spatial objects and to the chosen space-filling curve:

$$gap_{est}(p_k) = \frac{1}{\mu} \cdot \left(1 - \frac{1}{\mu}\right)^k, k \geq 0.$$

For typical Z-ordered GIS objects, for example, we observed that $\mu = 2.3$ seems to be a good choice. Between two base points, we assign fractions of the cardinality of the larger base point to model the observed decreasing frequency of the corresponding tile shapes:

$$gap_{est}(2^i \cdot p_k) = \frac{1}{2^i} \cdot gap_{est}(p_{k+1}), k \geq 0, 1 \leq i < d.$$

In most cases, gap_{est} is an accurate estimation of the real distribution of fractal gaps among a fine-grained interval sequence F . It can be used instead of $avgGap(\phi_i, F)$ to get better distributions for $BGaps_{right}(\phi_i, F)$ and $BGaps_{left}(\phi_i, F)$.

3 Empirical Evaluation

3.1 Experimental Setup

We implemented the proposed functions for the estimation of selectivity and execution cost on the Oracle Server Release 8.1.6 using built-in methods for statistics collection, analytic SQL functions, and the PL/SQL procedural runtime environment. All experiments were performed on an Athlon/750 machine with IDE hard drives. The database block cache was set to 500 disk blocks with a block size of 8 KB and was used exclusively by one active session. The experiments for the evaluation of statistics, selectivity estimation, and cost model have been executed on Z-ordered interval sequences representing the polygons of the SEQUOIA 2000 benchmark with an approximation

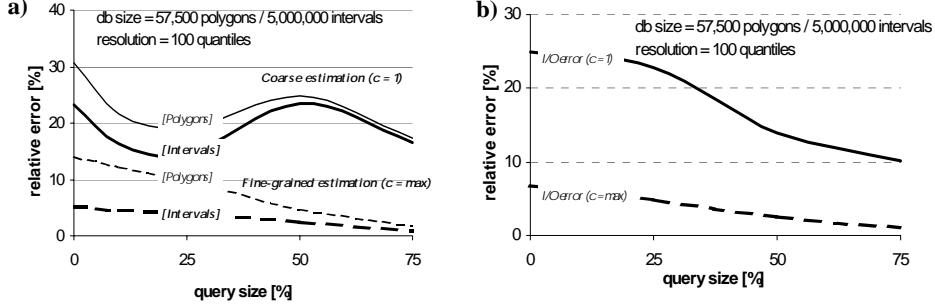


Figure 2: Relative error of **a)** selectivity estimation **b)** cost estimation

error of 6.3%. The following results show the averages of, in total, 100 intersection queries.

We have evaluated the average relative error w.r.t. the query size, i.e. the percentage of the data space covered by the query region. Figure 2a shows that for a coarseness of $c = 1$, we get relative selectivity estimation errors between 15% and 32%. Thus, even with the maximal aggregation, the computed estimate gives the query optimizer a good impression of the spatial selectivity. The quality of this hint improves with increasing c , as for $c = \max$ (i.e. $c = f$) in the experiment. For the actual query results we measured both selectivities: with respect to the total number of stored intervals, and with respect to the stored interval sequences, i.e. polygons. Note that the relative error to the actual polygon-based selectivity (*Polygons*) is roughly in the order of the relative error to the actual interval-based selectivity (*Intervals*). Thus, the selectivity on the single intervals largely reflects the selectivity on the original spatial objects. Nevertheless, what we have to provide as input for the cost estimation is the more accurate interval-based estimate.

We used the estimated selectivity of the previous paragraph as input for the I/O cost model. The extensible query optimizer uses the resulting estimations to decide upon the usability of the RI-tree for specific queries. Figure 2b depicts the corresponding results for a coarseness of $c = 1$ and $c = \max$. The I/O error for $c = 1$ at a query size near 0% averages 24.9% and decreases to 10.2% at 75% query size.

Figure 3 compares the absolute estimations and the actual cost for the blocked output of results ($output_{I/O}$). In addition, $join_{I/O}$ denotes the overhead due to the nested-loop join with the transient query tables. For the sake of comparability to the analytical I/O complexity, the results are shown with respect to the actual query selectivity. Our interpretation of these results is twofold: First, the real I/O cost show that the total I/O is largely determined by the cardinality of the query result, whereas the overhead for the join processing remains almost constant. The relative cost of the join overhead decreases from 100% at 0% selectivity to almost 0% at 100% selectivity. According to these empirical results, the overhead of $join_{I/O}$ is negligible for higher values of the query selectivity. Second, we observe that our cost model not only yields tight estimations for the total query cost, but also reflects the distribution between the output and join cost rather accurately. As expected, the accuracy of the cost estimation increases with a higher granularity c of the coarse interval sequence. Considering the comparable empirical results for cost-models on stand-alone R-trees [6], and the often significant difference to

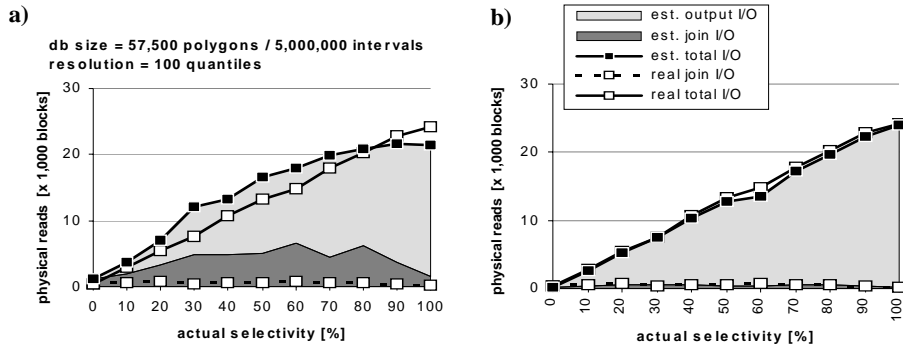


Figure 3: Output cost and join overhead for queries using coarse interval sequences with **a)** $c = 1$ and **b)** $c = \text{max}$.

the cost of alternative access paths including full-table scans, we conjecture that already a coarse estimation with $c = 1$ is well suited for spatial queries on the RI-tree. A fine-grained computation is, of course, much more accurate, but already anticipates a significant amount of the cost of the potential query. Regardless of the actual query selectivity, the cost computation took about 0.05 seconds for $c = 1$.

4 Conclusions

In this paper, we proposed a model to estimate the selectivity and the I/O cost of interval sequence intersection queries. According to our experimental evaluation and the comparable results of known cost models for other access methods, the computed estimations are accurate enough to give the optimizer an impression of the potential cost if the RI-tree is included into the execution plan.

References

1. Kriegel H.-P., Pfeifle M., Pötke M., Seidl T.: *A Cost Model for Interval Sequence Intersection Queries on RI-Trees*. Technical Report, University of Munich, 2003.
2. Kriegel H.-P., Pfeifle M., Pötke M., Seidl T.: *A Cost Model for Interval Intersection Queries on RI-Trees*. Proc. 14th Int. Conf. on Scientific and Statistical Database Management (SSDBM), 131-141, 2002.
3. Kriegel H.-P., Pötke M., Seidl T.: *Managing Intervals Efficiently in Object-Relational Databases*. Proc. 26th Int. Conf. on Very Large Databases (VLDB), 407-418, 2000.
4. Kriegel H.-P., Pötke M., Seidl T.: *Interval Sequences: An Object-Relational Approach to Manage Spatial Data*. Proc. 7th Int. Symposium on Spatial and Temporal Databases (SSTD), LNCS 2121, 481-501, 2001.
5. Lomet D.: *B-tree Page Size When Caching is Considered*. ACM SIGMOD Record, 27(3), 28-32, 1998.
6. Theodoridis Y., Stefanakis E., Sellis T.: *Efficient Cost Models for Spatial Queries Using R-Trees*. IEEE Transactions on Knowledge and Data Engineering, 12(1), 19-32, 2000.