

Benchmarking Spatial Joins *À La Carte*

Oliver Günther* Vincent Oria† Philippe Picouet‡ Jean-Marc Saglio‡ Michel Scholl§

Abstract

Spatial joins are join operations that involve spatial data types and operators. Spatial access methods are often used to speed up the computation of spatial joins. This paper addresses the issue of benchmarking spatial join operations. For this purpose, we first present a WWW-based benchmark generator to produce sets of rectangles. Using a Web browser, experimenters can specify the number of rectangles in a sample, as well as the statistical distributions of their sizes, shapes, and locations. Second, using the generator and a well-defined set of statistical models we define several tests to compare the performance of three spatial join algorithms: nested loop, scan-and-index, and synchronized tree traversal. We also added a real-life data set from the Sequoia 2000 storage benchmark. Our results show that the relative performance of the different techniques mainly depends on two parameters: sample size, and selectivity of the join predicate. All of the statistical models and algorithms are available on the Web, which allows for easy verification and modification of our experiments.

1. Introduction

Spatial joins are join operations that involve spatial data types and operators. Examples include queries such as

- Find all houses that are located within 10 kilometers of a lake, or
- Find all fields that grow wheat and that belong to the Smith or the Jones property.

Houses, lakes, fields, and properties are represented by a relation or a class, respectively. *Within 10 kilometers of and belong to* are spatial predicates.

Copyright 1998 IEEE. Published in the Proceedings of SSDBM'98, July 1-3 1998 in Capri, Italy. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from the IEEE. Contact: Manager, Copyrights and Permissions / IEEE Service Center / 445 Hoes Lane / P.O. Box 1331 / Piscataway, NJ 08855-1331, USA. Telephone: + Intl. 732-562-3966.

Günther [7] gives the following definition of a spatial join in a relational context:

The spatial join of two relations R and S , denoted by $R \bowtie_{i\theta j} S$, is the set of tuples from $R \times S$ where the i -th column of R or the j -th column of S are of some spatial data type, θ is a binary spatial predicate, and $R.i$ stands in relation θ to $S.j$.

Typically, one dedicated column in each relation R and S is of some spatial data type, representing the spatial extension of the corresponding data object. We can then just write $R \bowtie_{\theta} S$ as a shorthand for the spatial join $R \bowtie_{i\theta j} S$, where i and j refer to those dedicated columns in R and S , respectively. For the spatial predicate θ , there are many possibilities, including

- *intersects*
- *contains*
- *enclosed_by*
- *distance λq , with $\lambda \in \{=, \leq, <, \geq, >\}$ and $q \in \mathbb{R}_0^+$*
- *northwest_of*
- *adjacent_to*

Intersection is perhaps the most important spatial predicate [6]. Nevertheless, the intersection join is just *one* type of spatial join, albeit an important one. Unfortunately, many papers use the terms intersection join and spatial join as synonyms, which can lead to misunderstandings. In particular, many algorithms have been presented only in the context of intersection joins; a generalization to other θ -predicates is not immediately obvious [4, 17, 9, 10, 11].

For the computation of spatial joins, one usually employs a two-step approach. In the *filter step* one works with approximations of the actual data objects in order to

*Institut für Wirtschaftsinformatik, Humboldt-Universität zu Berlin, Spandauer Str. 1, 10178 Berlin, Germany, guenther@wiwi.hu-berlin.de

†Department of Computing Science, University of Alberta, Edmonton, T6G 2H1, Canada, oria@cs.ualberta.ca

‡École Nationale Supérieure des Télécommunications, 46, Rue Barrault, 75634 Paris Cedex 13, France, {picouet|saglio}@inf.enst.fr

§CNAM and INRIA, Domaine de Voluceau, Rocquencourt, France, Michel.Scholl@inria.fr

reduce the number of object pairs to be investigated in detail. Minimum bounding boxes (MBBs), also called minimum bounding rectangles (MBRs), are a common method of approximation. For each object pair that passes the filter step, we proceed with a *refinement step* where we retrieve the exact spatial extensions of the data objects from disk and check the join predicate in detail. In this paper we are exclusively concerned with the filter step of the join computation. The cost of the refinement step is nearly identical for most common computation strategies, certainly for the ones we study here. One possible exception is the PBSM technique by Patel and DeWitt [17], which optimized the refinement step using a common computational geometry technique called *plane sweep*.

This paper addresses the issue of *benchmarking* spatial join operations. For this purpose, we first present a WWW-based tool to produce sets of rectangles *à la carte*. Experimenters can use a standard Web browser to specify the number of rectangles, as well as their distributions with regard to size, shape, and location. Various common statistical distributions are supported for that purpose. Second, using the rectangle generator and a well-defined set of statistical models we defined several tests to compare the performance of three spatial join algorithms: nested loop, scan-and-index, and synchronized tree traversal. We also added a real-life data set, the Sequoia 2000 storage benchmark [21].

One of the critical issues in benchmarking is to make the results of an experiment both verifiable and robust. *Verifiable* means that other researchers should be able to repeat experiments easily and come to similar conclusions. *Robust* means that the results should hold not only in the particular environment of the original experiment but in a more general setting as well. Moreover, it should be easy to integrate the algorithms and data sets of the experiments into other benchmark experiments by other researchers. Both criteria are rarely met in experimental computer science [22]. Our Web interface, which provides access to the complete set of algorithms and experiments, is an important step in this direction. Section 2 describes the rectangle generator we built for the purpose of this study. We also specify the statistical models we used for the subsequent performance analysis. In Section 3 we survey approaches to compute spatial joins and discuss results of previous performance comparisons. Section 4 presents the setup and the results of our experiments. Section 5 concludes with an outlook on future work.

2. The Benchmark

2.1. The Rectangle Generator

At the École Nationale Supérieure des Télécommunications (ENST) we have implemented a tool to generate sets of rectangles with edges parallel to the

C	coverage: the ratio between (i) the sum of the areas of all rectangles, and (ii) the area of the universe
x	x -coordinate of the rectangle's lower left hand corner
y	y -coordinate of the rectangle's lower left hand corner
t	inclination of the rectangle's main diagonal (to control its shape)
a	area of the rectangle
N	number of rectangles (sample size)
x_{min}	smallest possible x -coordinate
y_{min}	smallest possible y -coordinate
x_{max}	largest possible x -coordinate
y_{max}	largest possible y -coordinate
U	size of the universe: $(x_{max} - x_{min})(y_{max} - y_{min})$

Table 1. Parameters of the rectangle generator

axes. Users can specify the parameters listed in Table 1. The first five parameters are modeled by means of statistical distributions. We currently support the uniform distribution $\mathcal{U}(min, max)$, the normal distribution $\mathcal{N}(\mu, \sigma)$ and the exponential distribution $\mathcal{E}(\mu, min, max)$.

Dependencies between variables are taken into account by the interface. If one has specified, for example, the coverage C , the size of the universe U , and the sample size N , the mean area of the rectangles in the sample (μ_a) will be automatically instantiated as CU/N .

If a generated rectangle does not fit into the universe, it is discarded and a new rectangle is generated in its place. Given a sensible choice of parameters, in particular $\mu_a \ll U$, the effect of these heuristics on the distribution is marginal. Moreover, for $\mu_a \ll U$ it hardly matters whether the parameters x and y denote a rectangle's lower left hand corner or, for example, its centerpoint.

The rectangle generator is available on the World Wide Web at <http://www.enst.fr/bdtest/sigbench/menu.html>. Users can transmit parameters to the generator and obtain a corresponding random sample. Figures 8–10 in the appendix show the current Web interface. Each user-specified model (i.e., the choice of distributions and parameter values) is saved on the ENST server under a name that is sent back to the user, together with the sample. This way users can later refer to their models and use them in their benchmarks. Note that we do not store the samples but only the underlying statistical models.

$$\begin{aligned} x, y &\sim \mathcal{U}(0, 1) \\ t &\sim \mathcal{U}(0, \pi/2) \\ a &\sim \mathcal{N}(1/N, \sigma) \end{aligned}$$

Biotopes-N

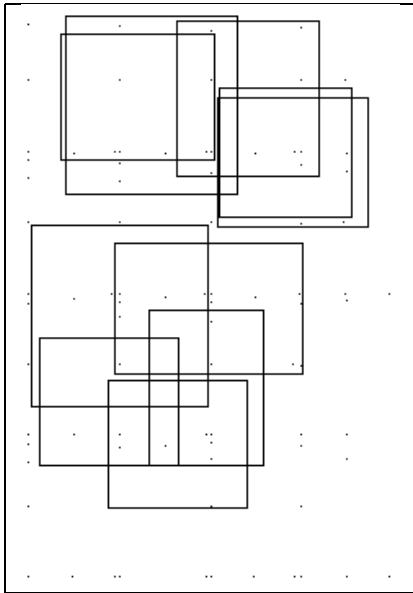


Figure 1. A sample of Biotopes-10

$$\begin{aligned} x, y &\sim \mathcal{U}(0, 1) \\ t &\sim \mathcal{N}(\pi/4, \sigma) \\ a &\sim \mathcal{E}(5 \times 10^{-2}/N, 4 \times 10^{-2}/N, 20/N) \end{aligned}$$

Cities-N

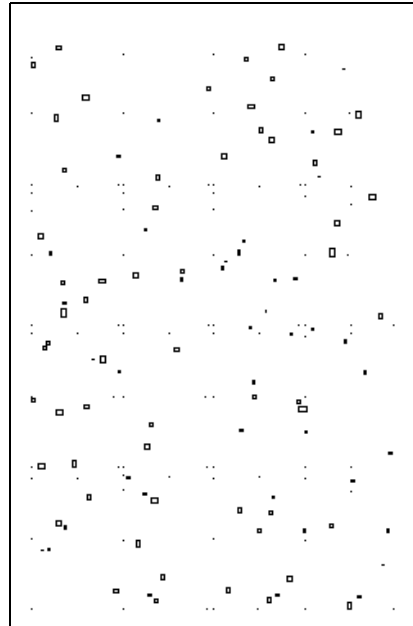


Figure 2. A sample of Cities-100

2.2. A typical Workload

Instead of combining different parameter constellations at random, we have defined three statistical models that simulate some typical cartographic applications.

The first model, called “Biotopes,” simulates a geological or biotope map. It contains relatively few large rectangles that are uniformly distributed in the universe. The coverage is 100%, which means that different formations may overlap but not to a large degree. The shape of the rectangles (expressed by the inclination of the rectangle’s main diagonal) is uniformly distributed. This situation can be modeled by the parameter configuration given in Fig. 1, which also pictures a sample for $N = 10$. Different values of N essentially lead to a change of scale: *Biotopes-100*, for example, contains 10 times as many objects as *Biotopes-10*. Their average area, however, is 10 times smaller.

The second model, called “Cities,” simulates the distribution of cities on a road map. The map contains many polygons of relatively small size. The polygons are uniformly distributed on the map. Coverage is 5%, which means that there is virtually no overlap. The shape is normally distributed around the square shape. Long and thin rectangles are rare. The parameters for this model, as well as a random sample for $N = 100$, are given in Fig. 2.

The third model simulates a world map. It is obtained by

nesting two submodels. In a first step, N_I relatively large rectangles are generated using the parameter constellation I given in Fig. 3. Coverage is 30%, which is comparable to the percentage of land on the earth surface. Overlap may occur but will be small. Each of those N_I “Continents” is filled with N_{II} objects each, generated according to parameter constellation II and scaled down to fit the size of the particular continent. Coverage is 100%, and the shape of the objects is normally distributed around the square. As a result, there are $N_I \times N_{II}$ rectangles in this model, equally divided among N_I rectangular clusters. Fig. 3 gives a sample for $N_I = 10$ and $N_{II} = 100$. (In the figure, in each continent, only 10 out of the 100 smaller objects are visualized.)

To complement these three statistical models, we added two real-life samples of rectangles borrowed from the Sequoia 2000 storage benchmark [21]. Fig. 4 displays one of these samples and shows the skewed distribution of the objects in the plane.

3. Computation of Spatial Joins

To compute a classical (i.e., non-spatial) relational join $R \bowtie S$ efficiently, there are several well-known strategies, most notably nested loop, sort-merge, scan-and-index, hash

$x_I, y_I \sim \mathcal{U}(0, 1)$ $t_I \sim \mathcal{N}(\pi/4, \sigma)$ $a_I \sim \mathcal{N}(0.3/N_I, \sigma)$
$x_{II}, y_{II} \sim \mathcal{U}(0, 1)$ $t_{II} \sim \mathcal{N}(\pi/4, \sigma)$ $a_{II} \sim \mathcal{N}(1/N_{II}, \sigma)$

Continents- N_I - N_{II}

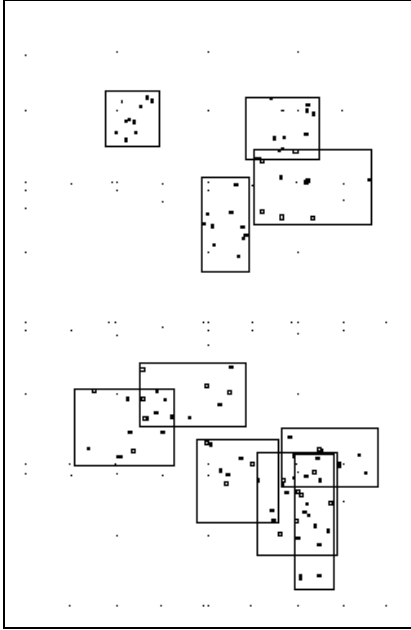


Figure 3. A sample of Continents-10-100

join, and join indices [14]. The application of these techniques to spatial joins is not always straightforward. We discuss the various approaches in turn.

3.1. Nested Loop

The simple *nested loop* approach compares each tuple in R with each tuple in S . Its performance is proportional to the product of the sizes of R and S , $|R| \cdot |S|$. Of course, this basic strategy also works for spatial joins. However, its lack of efficiency with larger data sets becomes even more obvious in the case of spatial data, where predicates are usually much harder to compute than simple comparison predicates on real numbers.

3.2. Sort-Merge

If the relations R and S can be sorted according to the tuple values in columns i and j , respectively, and if θ is a simple comparison predicate, such as $=$, $>$, or \leq , then there are more efficient ways to compute a join. The *sort-merge* strategy first sorts R on column i and S on column

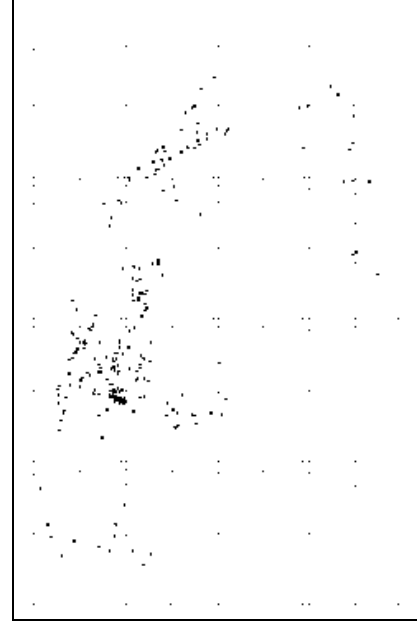


Figure 4. Sequoia-16: a real-life model

j . Then R and S are merged and checked for matching tuples. The running time of this algorithm is proportional to $|R| \log |R| + |S| \log |S| + |J|$, where $|J|$ is the cardinality of the result of the join.

In the case of spatial joins, however, sort-merge often does not work because *there is no total ordering among spatial objects that preserves spatial proximity*. As a result, for many θ -predicates there is no sort that makes sure that one catches all matching tuples during the following merge. For an example, consider Fig. 5, where the space is divided into square cells by means of a grid. The cells are sorted in *Peano order* (also called *locational codes* or *z-ordering* [15]), a common way of spatial sorting. Let θ be *adjacent*, let the relation R contain the cells 1, 3, and 4, and let S contain the cells 2, 7, 8, and 9. With sort-merge, one first sorts R into the sequence (1, 3, 4) and S into the sequence (2, 7, 8, 9). During the merge, one obtains in sequence the matching pairs (1, 2), (4, 2) and (4, 7). The matching pair (3, 9) remains undetected. Similar examples can be constructed for any other spatial ordering.

One notable exception from this effect is the θ -predicate *intersects*, for which sort-merge strategies can be used rather efficiently. One possible implementation based on Peano ordering has been described by Orenstein [15]. Abel et al. [1] later extended this work to support spatial join processing in a *distributed* environment. Becker et al. store the bounding boxes of the spatial objects as points in a higher dimension and use a grid file to find matching pairs [2]. Another approach is to take advantage of the plane-sweep tech-

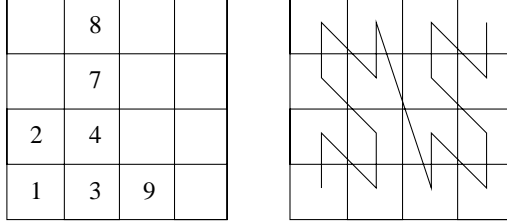


Figure 5. A spatial grid with the corresponding Peano sequence

nique known from computational geometry [18]. Rotem [19] uses this technique to build a spatial join index from existing grid files. Patel and DeWitt [17] partition the universe into tiles and use plane-sweep to find matching tuples in each tile.

3.3. Scan-and-Index

Another approach that takes advantage of the sortability of the columns involved is the *scan-and-index* strategy (also called *index-supported joins*). This approach can be applied if at least one of the relations involved (say R) has an index defined on the relevant column i that supports the join operator θ , i.e., the retrieval of matching tuples. A typical example would be a relation with a B^+ -tree on column i and θ being a simple comparison predicate ($=, <, \leq, >, \geq$). In that case one may scan the other relation (say S) and use the index on R to find the matching tuples for each tuple in S . If an index search takes time $\log |R|$, this algorithm results in a performance proportional to $|S| \log |R| + |J|$. This strategy can easily be adapted to spatial joins, provided there exists a suitable spatial index on one or more of the relations involved.

3.4. Hash Joins

For equality joins, an efficient approach is to hash both input relations with the same hash function on the join attribute (partition phase), and then to join the buckets in a pairwise manner (join phase). This *hash join* technique has some problems when applied to spatial joins because there are no equivalence classes as in the case of equality. Lo and Ravishankar [11] provide an interesting solution to this problem. There are two crucial differences in comparison to the non-spatial case. On the one hand, a data item may be “hashed” into multiple buckets. On the other hand, the hash (or partition) function for the two input relations may differ. Their experiments show that the spatial hash-join is highly competitive.

$o_1 \theta o_2$	$o'_1 \Theta o'_2$
o_1 within distance d from o_2 (measured between centerpoints)	o'_1 within distance d from o'_2 (measured between closest points)
o_1 intersects o_2	o'_1 intersects o'_2
o_1 includes o_2	o'_1 intersects o'_2
o_1 contained in o_2	o'_1 intersects o'_2
o_1 northwest of o_2 (measured between centerpoints)	o'_1 intersects the NW quadrant formed by the right vertical and the lower horizontal tangent of o'_2
o_1 reachable from o_2 in x minutes	o'_1 intersects the x -minute buffer of o'_2

Table 2. θ - and corresponding Θ -predicates.

3.5. Synchronized Tree Traversal

If hierarchical indices are available on both input relations, the scan-and-index technique can be extended in such a way that both indices are searched depth-first in a synchronized manner, with the two depth-first searches being guided by hints from each other. This technique has no immediate equivalent in traditional join processing.

Günther [7] proposed an algorithm based on the fact that many indices organize the data objects and bucket regions into a *PART-OF hierarchy*. Except for the root, each node's corresponding bucket region is completely contained in the bucket region corresponding to its parent node. Typical examples for this class of indices (also called *generalization trees*) are the R-tree [8] and the R*-tree [3]. While overlaps between bucket regions at the same tree level are forbidden in some of those index structures (e.g., the R⁺-tree), they do not pose a problem for the following join algorithm.

The idea of the algorithm is to examine higher levels of the tree first to see which branches may contain data objects that are of interest to the join to be computed. For that purpose, it is useful to define a predicate Θ , such that for two bucket regions o'_1 and o'_2 , $o'_1 \Theta o'_2$ is true if the corresponding subtrees may contain data objects o_1 and o_2 , respectively, such that $o_1 \theta o_2$. In that case it is necessary to go down the subtrees and investigate the situation at a finer granularity. In order to be an efficient filter, Θ -predicates should be both selective and relatively easy to compute. Table 2 gives several examples; note that the chosen Θ -predicates are often similar or identical to the corresponding θ -predicates.

Now let $GT_{R,A}$ and $GT_{S,B}$ denote the generalization trees defined on the relevant spatial columns A and B of relations R and S , respectively. In order to compute the spatial join $R \bowtie_{A \theta B} S$, one first checks whether the two roots match, i.e., whether $root(GT_{S,A}) \Theta root(GT_{R,B})$. If no, the search terminates; there are no matching tuples. If yes, we continue by checking for which children a' of $root(GT_{R,A})$ and b' of $root(GT_{S,B})$ the condition $a' \Theta b'$ is true. For each qualifying pair one appends an entry (a', b') to the queue

QualPairs. For each tuple in *QualPairs*, one proceeds recursively until all matching tuples have been found.

For the special case of the tree structure being an R-tree and θ meaning *intersects*, Brinkhoff et al. have independently proposed and implemented an efficient version of this algorithm [4]. When one of the input relations to the intersection join does not have an R-tree already available, Lo and Ravishankar [9] propose building a tree index on the fly. The index, called *seeded tree*, is similar to an R-tree but is allowed to be unbalanced. In [10] the authors extend this technique to the case where none of the two input relations has a tree index available. The application of R-trees to predicates other than *intersects* has been discussed by Padias et al. [16].

3.6 Join Indices

If the database does not encounter too many updates, it is usually worthwhile to precompute the result of frequent joins and store it in a *join index* [23]. Join indices can be used for spatial joins although they lose some of their efficiency in that case [19, 12, 13]. First, updates become even more expensive because the computations involved are more complicated. Second, the efficient implementation of join indices, as described by Valduriez [23], relies on an ordering along the join attributes, which cannot be maintained in the spatial case.

4. Results of the Comparative Study

In our practical experiments we evaluated the following three algorithms to perform a spatial join.

- Nested Loop (NL);
- Scan-and-Index (SI);
- Synchronized Tree Traversal (STT).

As a testing environment, we chose the object-oriented database system O_2 [5]. All algorithms were implemented under O_2 version 4.5 and Sun OS 4.1.3 on a Sparc station Sun System 10.

For SI and STT, we used an efficient secondary-memory implementation of a special quadtree data structure [20]. The approach relies on z-ordered quadtree-indexed relations. Each z-ordered index is mapped onto the system's B^+ -tree in order to take advantage of its clustering mechanism. This technique provides more flexibility and a simpler design than an index implemented in the system's kernel, without compromising too much on performance.

During initial tests, we distinguished the case where the two input samples are mapped onto the *same* universe from the case where they are shifted against each other. In the

former case, the quadtree grids used for indexing the two samples are identical, otherwise they are different. The performance differences between these two cases, however, proved to be negligible. We consequently decided to drop this aspect from further consideration.

We initially concentrated on two θ -predicates: *intersects* and *northwest*. While for *intersects* the matching probability for two rectangles chosen at random is directly related to their distance, this does not matter for *northwest*. In fact, the probability that a randomly chosen rectangle r_1 is *northwest* of another randomly chosen rectangle r_2 is 25%, no matter where these rectangles are located (cf. the definition in Table 2). This means that for this predicate the expected result size is very large: in the average, 25% of the tuples in $R \times S$ qualify.

We performed 12 tests, each defined by the θ -predicate and the models underlying the two input relations. Table 3 gives an overview. Most tests compare synthetic models of different sample sizes. We also investigate two samples from the Sequoia 2000 storage benchmark. The first sample (*Sequoia-11*) contains 7972 rectangles, the second one (*Sequoia-16*) 971 rectangles.

Sample 1	Sample 2	Combined Sample Size
Biotopes-100	Biotopes-100	10^4
Biotopes-100	Cities-1,000	10^5
Cities-1,000	Cities-1,000	10^6
Continents-10-100	Continents-10-1000	10^6
Sequoia-16	Sequoia-11	7,740,812
Biotopes-1,000	Cities-10,000	10^7

Table 3. Test suite

For the experiments, we first generated three random samples for each test (i.e., for each line of Table 3). We then ran each of the three algorithms against the three random samples, resulting in nine runs per test. The numbers reported below are averages taken over the three runs corresponding to a given test-algorithm combination. The variance between any such three runs was negligible in all cases.

Figure 6 plots the performance gains of SI and STT, where gain is defined as the ratio of NL elapsed time over SI/STT elapsed time. Gain is plotted versus combined sample size, measured by the numbers of tuples in the Cartesian product, $|\text{Sample 1}| \times |\text{Sample 2}|$.

For the *intersects* operator, both SI and STT provided significant performance improvements compared to the nested loop strategy NL. Gains are between 2 and 100, increasing with larger sample sizes. SI seems to do somewhat better than STT for smaller sample sizes, whereas STT takes over for combined sample sizes larger than 1 million.

For *northwest* we obtain a different picture. In all of our tests, NL was the most efficient strategy, i.e., gain was

less than 1. The overhead associated with indices apparently outweighed any performance improvements gained from using them. This is because the *northwest* join usually returns a large number of tuples. There are no significant differences between SI and STT. Moreover, relative performance does not seem to depend on sample size anymore.

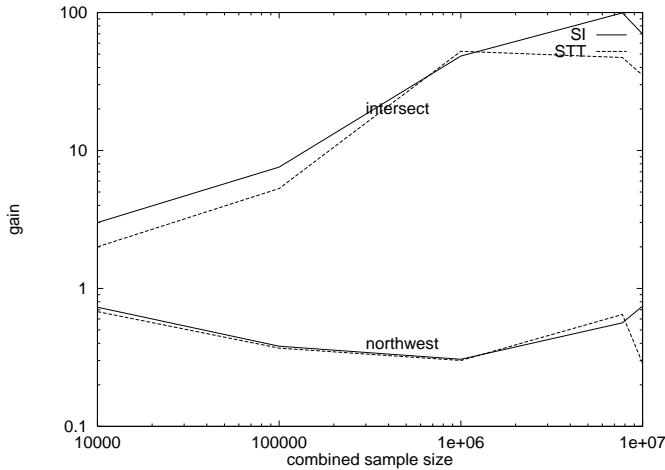


Figure 6. Relative performance NL/SI and NL/STT vs. combined sample size

An interesting and somewhat unexpected result was that the choice of model (i.e., “Biotopes” vs. “Cities” vs. “Continents”) had a relatively small impact on the measurements. For example, the test results for (*intersects*, *Cities*-1,000, *Cities*-1,000) and (*intersects*, *Continents*-10-100, *Continents*-10-1000) were almost identical. Additional tests not reported here confirmed this insight.

The use of real-world data versus synthetic data, on the other hand, seems to have a somewhat greater impact. As indicated by the peaks at $x = 7, 740, 812$, the Sequoia data is somewhat more complex to process for NL than the synthetic data sets. For *intersects*, the gains of SI and STT vs. NL are up to 50% higher for the Sequoia data sets than the hypothetical performance for a synthetic data set of the same size (obtained by linear interpolation). NL seems to suffer more from the non-uniformities of the Sequoia data set because they make caching less efficient. As NL is highly dependent on an efficient caching strategy, this leads to greater performance losses than in the case of SI and STT. For *northwest* the results are somewhat less conclusive although NL still seems to suffer more than STT.

The connections between performance gain, sample sizes, and θ -operators become clearer once one considers the performance behavior as a function of the matching probability or selectivity of the join predicate. Matching

probability¹ is defined as the ratio between the number of tuples retrieved and the combined sample size $|\text{Sample 1}| \times |\text{Sample 2}|$. We complement the tests listed in Table 3 by tests for the θ -operators *includes* and *contains*. Matching probability is highest for *northwest*, followed by *intersect*, *contains*, and *includes*, in that order.

As Figure 7 shows, matching probability is an excellent indicator of the observed variations in relative performance. Larger matching probabilities generally tend to lower the performance advantage expected from using an index. For large matching probabilities, the use of indices is no longer worthwhile because the associated overhead outweighs any potential performance gain.

For a given matching probability, there may still be up to one order of magnitude of difference in relative performance. As noted previously, these differences can be explained in terms of sample size. Larger sample sizes lead to larger performance advantages for both index-based strategies SI and STT.

In summary, there are only two parameters that really seem to matter: matching probability (selectivity) and sample size. Other factors like the choice of model, the spatial distribution, or the overlap of the data objects did not seem to have a major impact. This could be regarded as a positive result; it means that query optimizers can concentrate on those two simple parameters without getting involved with other specifics of the given data sets.

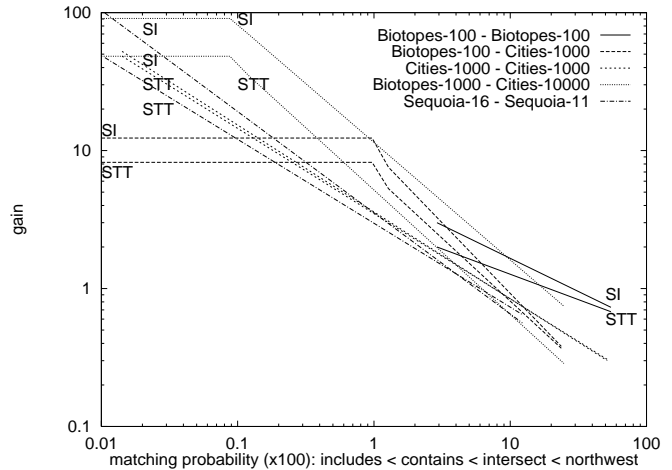


Figure 7. Relative performance SI/NL and STT/NL vs. matching probability

Our tests may be validated through the World Wide Web. All of our algorithm implementations are available through

¹We prefer the term *matching probability* over the more common *selectivity* because of the confusing fact that a high selectivity corresponds to a low number of tuples retrieved, and vice versa.

<http://www.enst.fr/bdtest/sigbench/menu.html>. The form-based interface shown in Fig. 8–10 allows users to create samples of the three models described above and to use them as inputs to these programs. Moreover, users can refer to models they specified previously with our rectangle generator.

5. Conclusions

In this paper we investigated the issue of *benchmarking* spatial join operations. Our first contribution is a WWW-based tool to produce random benchmarks for a given sample size and distribution. Experimenters can use a standard Web browser to specify the number of rectangles they want in a sample, as well as distributions of their sizes, shapes, and locations. Various statistical distributions are supported for that purpose.

Our second contribution is a performance evaluation of several common algorithms to compute a spatial join. With the help of the rectangle generator, we defined several tests and ran experiments to compare the nested loop strategy with two index-based strategies: scan-and-index, and synchronized tree traversal. Our results showed that the relative performance of the two index-based strategies compared to NL mainly depends on two parameters: matching probability (selectivity) and sample size. As expected, smaller matching probabilities clearly favor the index-based strategies. Large matching probabilities, on the other hand, render these strategies virtually worthless compared to the simple nested loop strategy, because the associated overhead outweighs any potential performance advantages. As for sample size, larger samples are more advantageous for the index-based strategies, simply because of their lower time complexities. In comparison to matching probability, however, the impact of sample size is generally much smaller. The same is true for the choice of model, which did not have a significant impact except when one compares real-life data with synthetic samples. Real-life data seems to be a major problem for NL because it affects the caching efficiency considerably. Index-based strategies are less affected by this. Finally, the difference between the two index-based strategies was negligible in comparison to the other effects we observed.

A later implementation will also include an evaluation of other join strategies. Sort-merge and hash joins are currently being implemented. We also plan to enhance the rectangle generator to support a greater variety of distributions, such as skewed distributions or correlative x-y-distributions. Our long-term objective is to bring to the community statistically well founded workloads sufficient for a variety of benchmarking applications.

Our Web interface, which provides access to the complete set of algorithms and experiments, is an important

step to make the results of our evaluation both verifiable and robust. Other researchers should be able to repeat our experiments easily and come to similar conclusions. Our results should hold not only in the particular environment of the original experiment but also in a more general setting. Moreover, it should be easy to integrate the algorithms and data sets of the experiments into other benchmark experiments by other researchers. We invite the reader to access our Web site and do so.

References

- [1] D. J. Abel, B. C. Ooi, K.-L. Tan, R. Power, and J. X. Yu. Spatial joins in distributed spatial query processing. In *Advances in Spatial Databases*, number 951 in LNCS. Springer-Verlag, 1995.
- [2] L. Becker, K. Hinrichs, and U. Finke. A new algorithm for computing joins with grid files. In *Proc. IEEE 9th Int. Conference on Data Engineering*, 1993.
- [3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 322–331, 1990.
- [4] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using R-trees. In *Proc. ACM SIGMOD Conference on the Management of Data*, pages 237–246, 1993.
- [5] O. Deux et al. The story of O₂. *IEEE Trans. on Knowledge and Data Engineering*, 2(1):91–108, March 1990.
- [6] V. Gaede and W.-F. Riekert. Spatial access methods and query processing in the object-oriented GIS GODOT. In *Proc. of the AGDM'94 Workshop*, pages 40–52, Delft, The Netherlands, 1994. Netherlands Geodetic Commission.
- [7] O. Günther. Efficient computation of spatial joins. In *Proc. IEEE 9th Int. Conference on Data Engineering*, 1993.
- [8] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 47–54, 1984.
- [9] M.-L. Lo and C. V. Ravishankar. Spatial joins using seeded trees. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 209–220, 1994.
- [10] M.-L. Lo and C. V. Ravishankar. Generating seeded trees from data sets. In *Advances in Spatial Databases*, number 951 in LNCS. Springer-Verlag, 1995.
- [11] M.-L. Lo and C. V. Ravishankar. Spatial hash-joins. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 247–258, 1996.
- [12] W. Lu and J. Han. Distance-associated join indices for spatial range search. In *Proc. IEEE 8th Int. Conference on Data Engineering*, 1992.
- [13] W. S. Luk and W. Zhou. How spatial data models and DBMS platforms affect the performance of spatial join. In *Proc. 6th Int. Symposium on Spatial Data Handling*, 1994.
- [14] P. Mishra and M. H. Eich. Join processing in relational databases. *ACM Computing Surveys*, 24(1):63–113, March 1992.
- [15] J. A. Orenstein. Spatial query processing in an object-oriented database system. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 326–333, 1986.

[16] D. Papadias, Y. Theodoridis, T. Sellis, and M. J. Egenhofer. Topological relations in the world of minimum bounding rectangles: A study with R-trees. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 92–103, 1995.

[17] J. M. Patel and D. J. DeWitt. Partition based spatial-merge joins. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 259–270, 1996.

[18] F. P. Preparata and M. I. Shamos. *Computational geometry*. Springer-Verlag, New York, NY, 1985.

[19] D. Rotem. Spatial join indices. In *Proc. IEEE 7th Int. Conference on Data Engineering*, pages 10–18, 1991.

[20] M. Scholl, G. Grangeret, and X. Rehse. Point and window queries with linear spatial indices: An evaluation with o2. Technical Report RRC-96-09, Cedric Group, CNAM, Paris, 1996. ftp://ftp.cnam.fr/pub/CNAM/cedric/tech_reports/RRC-96-09.ps.Z.

[21] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The Sequoia 2000 storage benchmark. In *Proc. ACM SIGMOD Conference on Management of Data*, 1993.

[22] W. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software*, 28(1):9–18, 1995.

[23] P. Valduriez. Join indices. *ACM Trans. on Database Systems*, 12(2), 1987.

6. Appendix A: Related Web Pages

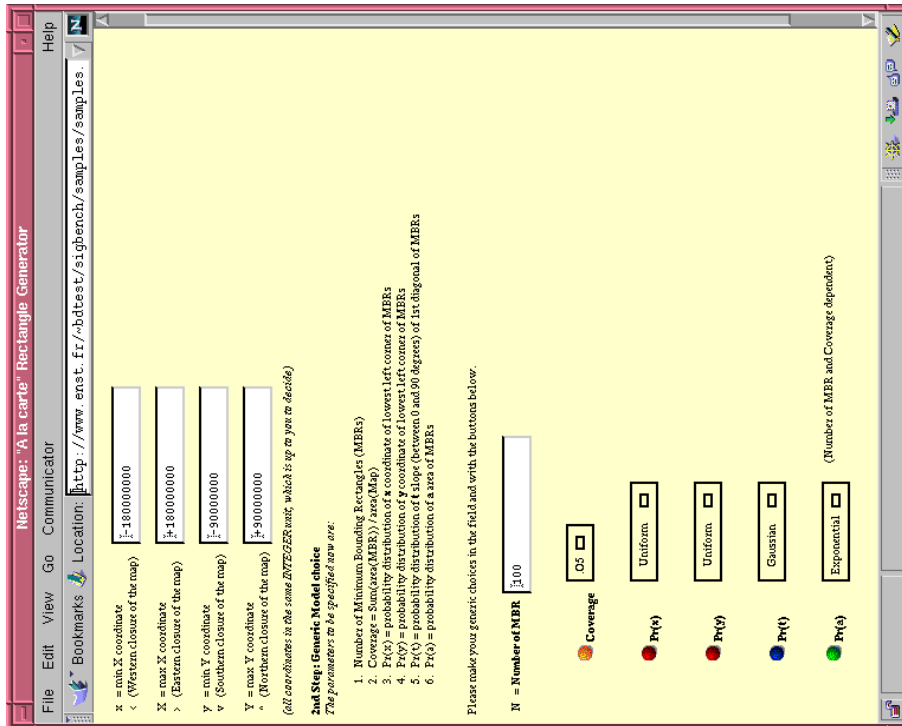


Figure 8. The Web interface to the rectangle generator (page 1)

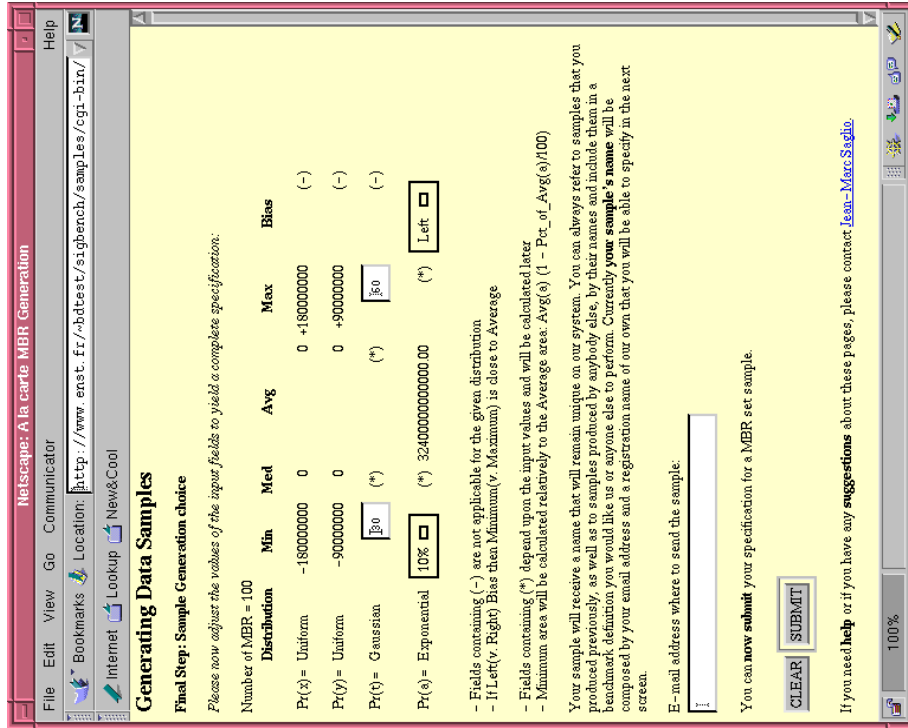


Figure 9. The Web interface to the rectangle generator (page 2)

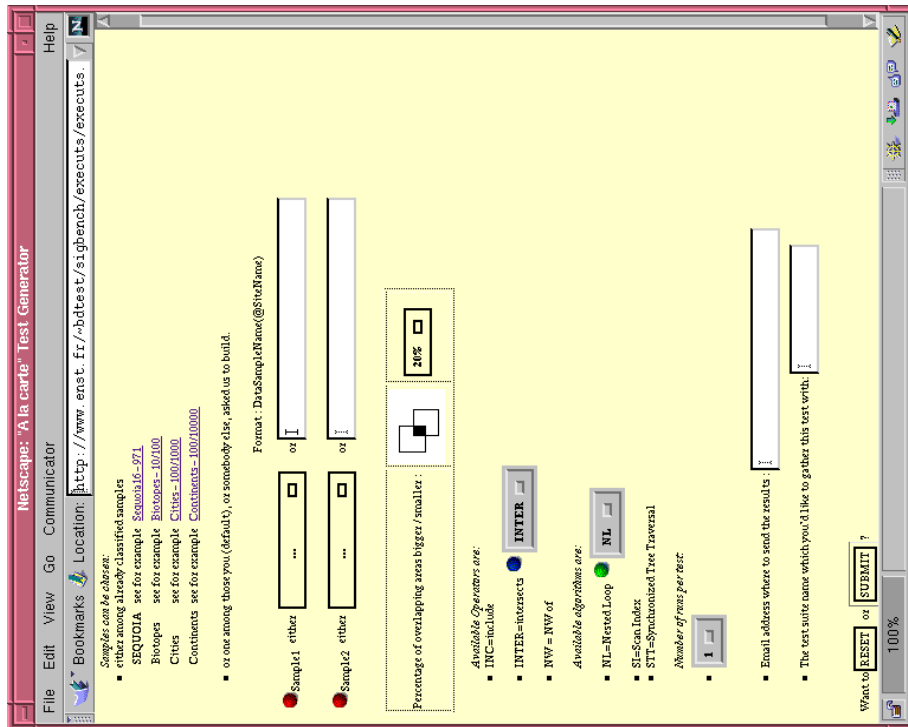


Figure 10. The Web interface to running experiments