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 tool to produce sets of rectangles. Experimentators can use a standard Web browser to specify the number of rectangles, as well as the statistical distributions of their sizes, shapes, and locations. Second, using the rectangle generator and a well-defined set of statistical models we defined several test suites 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. Our results confirm that the use of spatial indices leads to performance gains of several orders of magnitude. The tests also show that highly selective join predicates enjoy greater performance gains, and vice versa. 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 from 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 collections of objects that are each associated with a spatial location and a (usually non-zero) spatial extension. In a relational or object-oriented database system, these collections would typically each be represented by a relation or a class, respectively. *Within 10 kilometers from* and *belong to* are spatial predicates.

[*]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

Günther [Gün93] gives the following definition of the 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 and 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 a wide variety of possibilities, including

- $intersects(\cdot)$
- $contains(\cdot)$
- $enclosed_by(\cdot)$
- $distance(\cdot)\Theta q$, with $\Theta \in \{=, \leq, <, \geq, >\}$ and $q \in \Re_0^+$
- $northwest(\cdot)$
- $adjacent(\cdot)$

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 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, in the following *refinement step* 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 seems to be the PBSM technique by Patel and DeWitt [PD96] who optimized the refinement step using a common computational geometry technique called *plane sweep*.

For most θ-predicates, the intersection join $R \bowtie_{intersects} S$ plays an important role during the filter step [GR94]. 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 [BKS93, PD96, LR94, LR95, LR96].

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*. Experimentators 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 test suites 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 [SFGM93].

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 the

Parameter	Meaning
C	coverage: the ratio between the total area of all rectangles and 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	angle 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

Table 1: Parameters for the rectangle generator

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 [TLPH95]. 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 a variety of approaches to compute a spatial join 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 axes. Users can specify the parameters listed in Table 1. For the first five parameters, the user has to specify some statistical distribution with the usual parameters. We currently support the uniform distribution $\mathcal{U}(v, w)$ and the normal distribution $\mathcal{N}(\mu, \sigma)$.

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

If a generated rectangle does not fit into the universe given by $(X_{min}, Y_{min}, X_{max}, Y_{max})$, it is discarded and a new rectangle is generated in its place. Given a sensible choice of parameters ($\mu_a \ll 1; \mu_x, \mu_y \in \{0, 1\}$), the effect of these heuristics on the distribution is marginal.

The rectangle generator is available on the World Wide Web at URL http://www-inf.enst.fr/~bdtest/sigbench/menu.html. Anybody with Web access can transmit their parameters to the generator and obtain a corresponding random sample. Figures 8 and 9 in the appendix show the current Web interface. The models that a user specified (i.e., the parameterized distributions for the parameters) are saved on the ENST server under a name that is sent back to the user, together with the sample. On the ENST server, these two items are stored in a database, together with the corresponding parameter values and statistical distributions. Note that we do not store the samples but only the underlying statistical model. This way users can later refer to their models and use them in their benchmarks.

2.2 A typical workload

Instead of combining different parameter constellations at random, we have defined three statistical *models* that simulate certain 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 200%, which symbolizes that different formations may overlap but not to a large degree. The shape of the rectangles is uniformly distributed, i.e., the probability for a rectangle to be close to a square is equal to it being long and thin. This situation can be modelled by the parameter configuration given in Fig. 1, which also pictures a sample for $N = 10$. Different values of N basically lead to a change of scale. *Biotopes-100*, for example, contains 10 times as many objects as *Biotopes-10* but at with an area 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 the nesting of 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. To keep things simple, we have decided not to forbid overlap altogether. 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 objects have been visualized).

To these three statistical models, we have added two real life samples of rectangles borrowed from the Sequoia 2000 storage benchmark [SFGM93]. Fig. 4 displays one of these samples. Observe the sekwed distribution of the objects position in the plane.

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

Biotopes-N

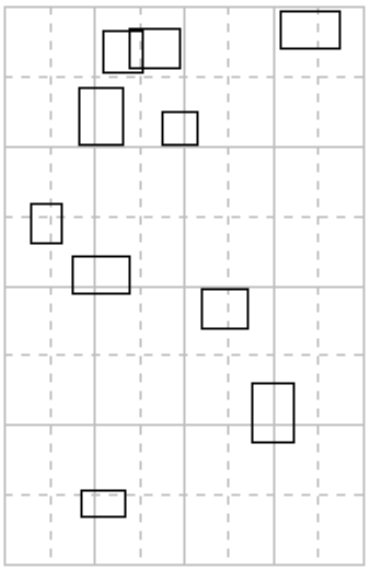

Figure 1: *Biotopes* model, N=10

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 join, and join indices [ME92]. 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 checks each tuple in R against each tuple in S whether there is a match. 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 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

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

Cities-N

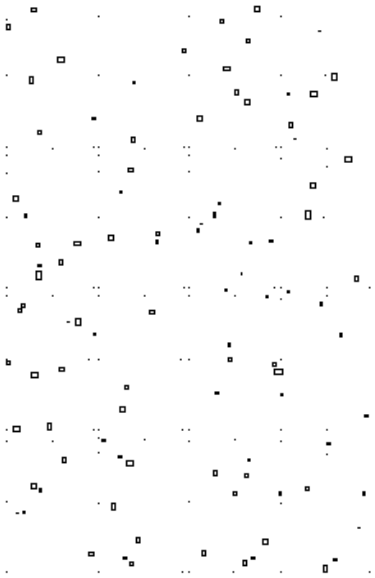

Figure 2: *Cities* model, N=100

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* [Ore86]), 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 [Ore86]. Abel et al. [AOT+95] 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 [BHF93]. Another approach is to take advantage of the plane-sweep technique known from computational geometry [PS85]. Rotem [Rot91] uses this technique to build a spatial join index from existing grid files. Patel and DeWitt [PD96] 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

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

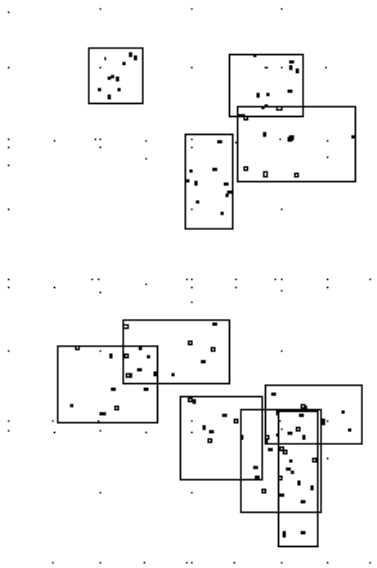

Figure 3: *Continents* model, N_I=10, N_{II}=100

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 on the join attribute, using the same hash function (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 [LR96] provide an interesting solution to this problem. There are two crucial differences in comparison to the relational 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.

3.5 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* [Val87]. A join index is a two-column relation that stores the tuple IDs of matching tuples. Each join then corresponds to a simple look-up in the join index relation followed by a retrieval of the tuples involved from the disk. If one only counts join computation time, this strategy is highly effective and beats all other strategies in most cases. On the other hand, it is often hard to forecast which joins are the most frequent. Furthermore, updates are not cheap, and in highly dynamic environments other strategies will catch up because their lower update costs outweigh the higher costs for join computation. Join indices can also be used for spatial joins although they lose some of their efficiency in that case [Rot91, LH92, LZ94]. First, updates become even more expensive because

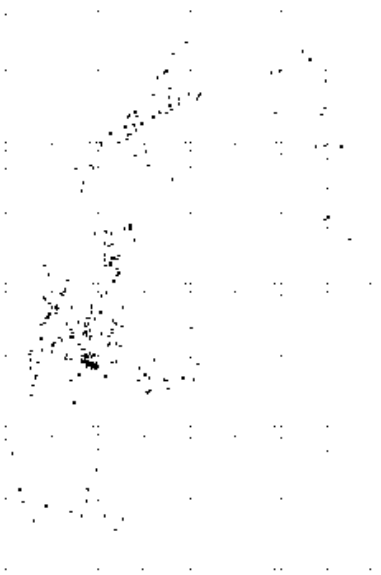

Figure 4: *Sequoia-16*: "real-life" model

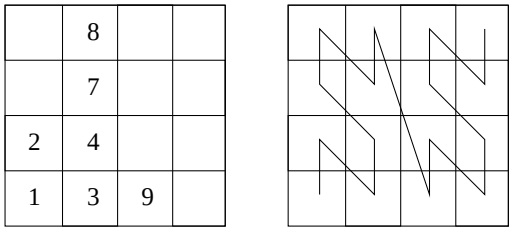

Figure 5: A spatial grid with the corresponding Peano sequence

the computations involved are more complicated. Second, the efficient implementation of join indices, as described by Valduriez [Val87], relies on an ordering along the join attributes, which cannot be maintained in the spatial case.

3.6 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 [Gün93] proposed an algorithm based on the fact that many indices organize the data objects and bucket regions into a *PART-OF hierarchy*. Typical examples for this class of indices (also called *generalization trees*), are the R-tree [Gut84] and the R*-tree [BKSS90]. Except for the root, each node's corresponding bucket region is completely contained in the bucket region corresponding to its parent node. While overlaps between bucket regions at the same tree level are forbidden in some of those index structures (such as the R^+-tree), they do not pose a problem for the following join algorithm.

$o_1 \theta o_2$	$o_1' \Theta o_2'$
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.

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. The semantics of a Θ-predicate are highly dependent on the semantics of the corresponding θ-predicate. Table 2 gives Θ-predicates for several common 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 $R \underset{A\theta B}{\bowtie} S$, 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 $root(GT_{R.A})$ the condition $a' \Theta root(GT_{S.B})$ is true, and for which children b' of $root(GT_{S.B})$ the condition $root(GT_{S.A}) \Theta b'$ is true. For each qualifying a' and each qualifying b' one appends an entry (a', b') to the list $QualPairs[1]$. For each tuple in $QualPairs[1]$, 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 [BKS93]. When one of the input relations to the intersection join does not have an R-tree already available, Lo and Ravishankar [LR94] propose building a tree index on the fly. The index, a so-called *seeded tree*, is similar to an R-tree but is allowed to be unbalanced. In [LR95] the authors extend this technique to the case where none of the two input relations has a tree index available. Abel et al. [AOT+95] present an R-tree-based approach to compute spatial joins in a *distributed* environment. The application of R-trees to predicates other than *intersects* has been discussed by Papadias et al. [PTSE95].

4 Results of the Comparative Study

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

θ-Predicate	Model 1	Model 2	Grid
intersects	Biotopes-100	Cities-1,000	same
	Biotopes-1,000	Cities-10,000	same
northwest	Biotopes-100	Cities-1,000	same

Table 3: Test suite 1

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

We chose for our experiment an O_2 DBMS platform [Deu89] on which the algorithms were implemented and run. The O_2 version 4.5 was run under 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 [Sam84] as an index developed at CNAM, Paris. The approach [SGR96] 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.

Two join predicates θ were mainly investigated: *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 matching 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. 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. An efficient query optimizer would therefore relegate this operation towards the end of the processing, where the number of tuples involved is already relatively small. We consequently dropped the large *continents* model from the measurements regarding this predicate.

Our current benchmark consists of three *test suites*. Each test suite consists of several *tests*, which are defined by the θ-predicate, the models underlying the two input relations, and the algorithm used. Moreover, we distinguish between the case where the two input samples are mapped onto the same universe, or shifted against each other by a random vector. In the former case, the grids used for indexing the two samples are identical, which simplifies processing considerably. In the latter case, the grids are different (unless one reorganizes one of the two samples).

The first test suite (Table 3) compares two *Cities* and two *Biotopes* models of different sample sizes. All samples are projected onto the same universe, i.e., the indexing grids are identical. Given the fact that we apply three algorithms to each combination of models, we obtain nine different tests.

θ-Predicate	Sample 1	Sample 2	Grid
intersects	Sequoia-16	Sequoia-11	same
northwest	Sequoia-16	Sequoia-11	same

Table 4: Test suite 2

θ-Predicate	Sample 1	Sample 2	Grid
intersects	Biotopes-100	Biotopes-100	different
	Cities-1,000	Cities-1,000	different
	Continents-10-1,000	Continents-10-1,000	different
northwest	Biotopes-100	Biotopes-100	different
	Cities-1,000	Cities-1,000	different

Table 5: Test suite 3

For the second test suite (Table 4), we added a real-life data set to the three synthetic models described above. The two input samples are drawn from the Sequoia 2000 Storage Benchmark [SFGM93]. The first sample *Sequoia-11*, contains 7972 rectangles, and the second one, *Sequoia-16*, 971 rectangles. Both samples are mapped onto the same universe, i.e., the underlying grids are identical. Given three algorithms, this second test suite consists of six different tests.

The third test suite (Table 5) compares two randomly generated samples of the *same* model, but shifted against each other by a random vector $V \sim \mathcal{U}((0,0),(1,1))$. This idea is applied to all three synthetic models and a variety of sample sizes. Because we also do not apply *northwest* to the *Continents* samples, we obtain 15 different tests for this third test suite.

Each test consists of several *rectangle generations* and *runs*. For each test (each line of a given table), there are different random drawings of the same statistical model. Furthermore, we performed several runs per test. The numbers used in the following evaluation are averages taken over the corresponding runs.

Figure 6 plots the performance gain 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.

For the *intersects* operator, both SI and STT provided significant performance improvements compared to the nested loop strategy NL. Both SI and STT lead to improvement factors (gain) between 2 and 100. Differences between SI and STT are inconclusive. The gain seems to increase with larger sample sizes.

For *northwest*, however, we obtain a somewhat different picture. In all of our tests, NL was the most efficient strategy. The overhead associated with the use of indices apparently outweighed any performance improvement gained from them. This is because join with NW predicate usually select a large number of couples. Once again, there are no significant differences between SI and STT. Moreover, relative performance does not seem to depend on sample size

anymore.

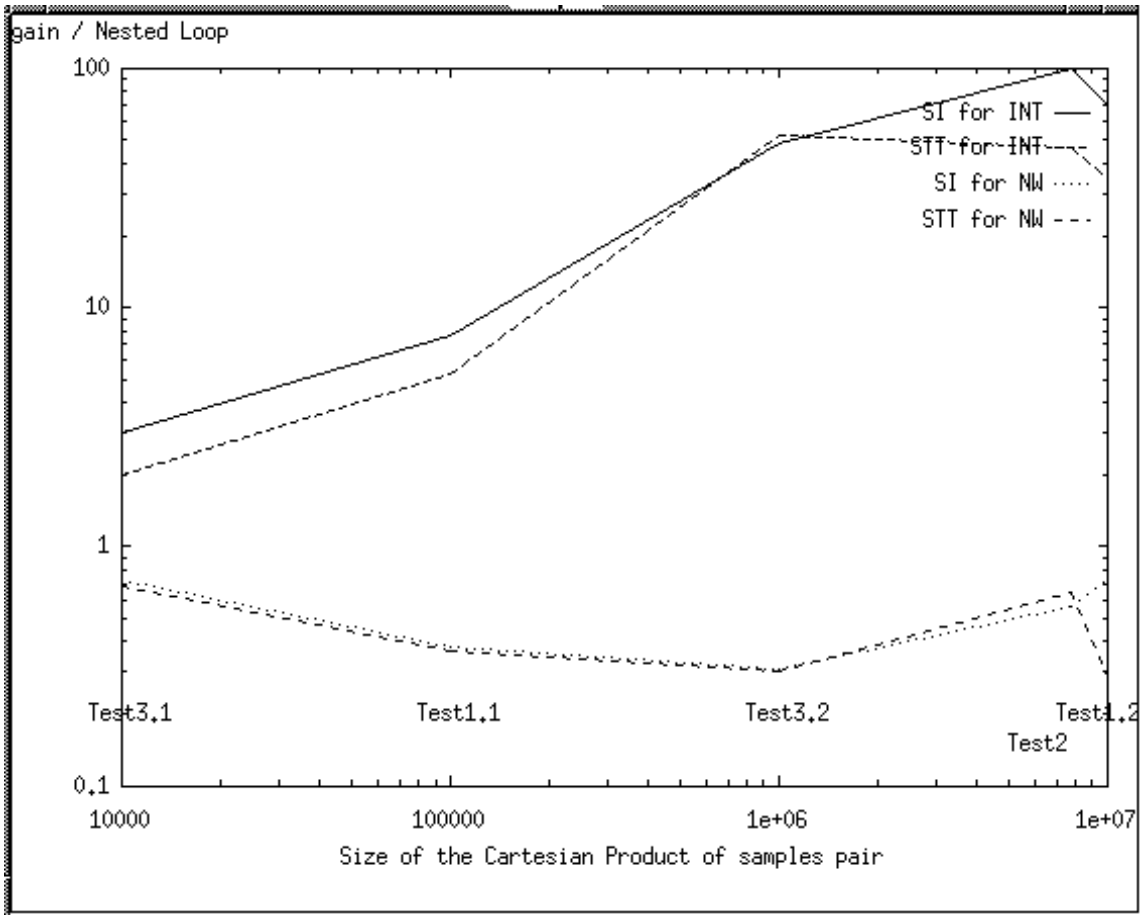

Figure 6: Relative performance SI/NL and STT/NL vs. combined sample size

The use of real-world data versus artificial data does not seem to make much of a difference. There is a slight increase in the gain of SI versus NL in the case of the Sequoia data but it seems hardly significant. In any case, it does not have an impact on the main insights obtained from our experiments.

The connections between performance gain, sample sizes, and θ-operators become clearer once one considers performance behavior versus selectivity. Selectivity is defined as the ratio between the number of tuples retrieved and the number of tuples in the Cartesian product. Not surprisingly, our tests revealed a much larger selectivity for *northwest* than for *intersect*. The use of different grids increased selectivities for *northwest* even further, simply because the shift increases the probability for two given data objects to be in a *northwest* relationship. For the *intersect* operator, selectivity tended to decrease for larger sample sizes. No such effect was observed for *northwest*.

As Figure 7 shows, selectivity is an excellent indicator of the observed variations in relative performance. Larger selectivities generally tend to lower the performance advantage expected from using an index. For large selectivities, the use of indices is no longer worthwhile because the associated overhead outweighs any potential performance gain. A join with *NW* usually

corresponds to a large selectivity. Then NL is preferable for such joins.

For a given selectivity, 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. This is not surprising if one looks at the complexities of the various algorithms. NL's complexity is linear in the combined sample size $|R| \cdot |S|$. In contrast, the complexity of SI is $|S| \log |R| + |J|$, where S is the relation to be scanned (usually the smaller one), and J is the set of qualifying tuples. STT, finally, has a complexity of only $\log |S| \log |R| + |J|$, due to the logarithmic complexity of the corresponding tree searches.

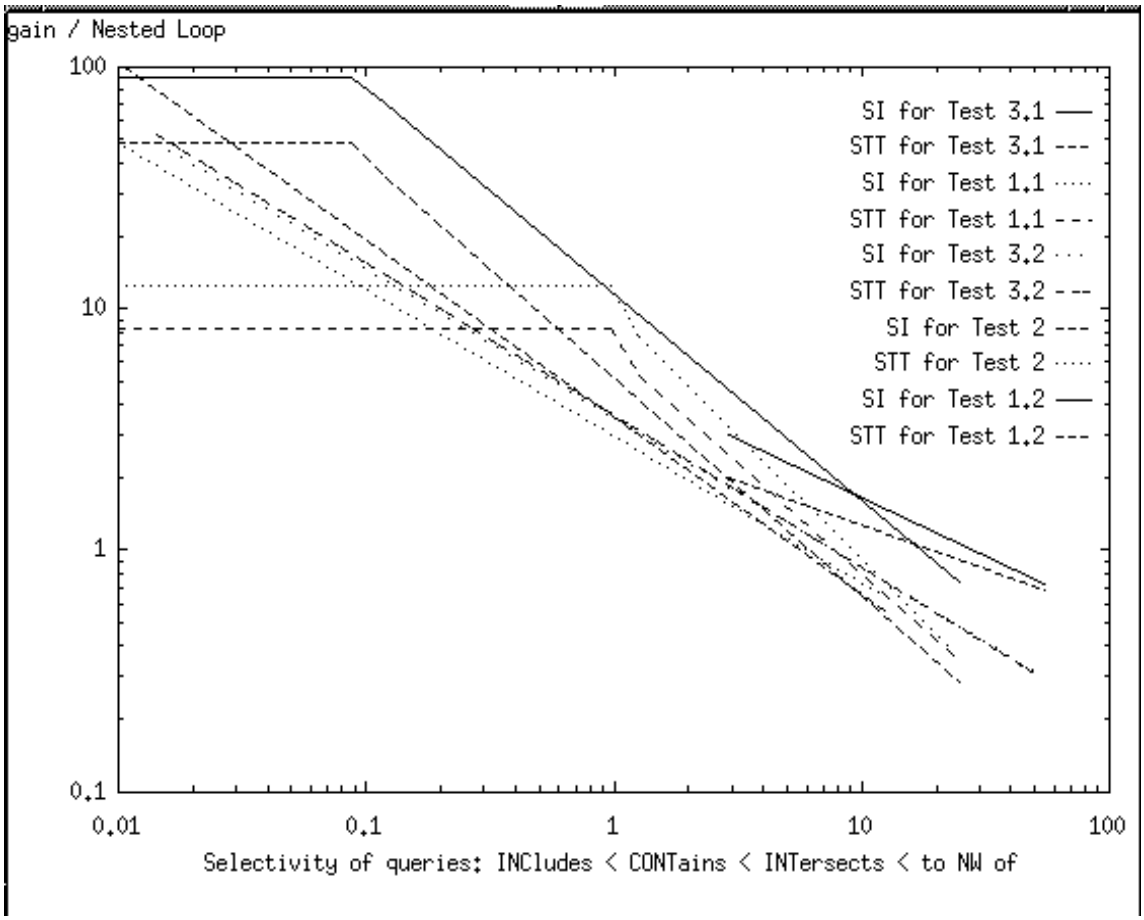

Figure 7: Relative performance SI/NL and STT/NL vs. selectivity

Our tests may be validated through the World Wide Web. All of our algorithm implementations are available through http://www-inf.enst.fr/~bdtest/sigbench/menu.html. The form-based interface shown in Fig. 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 (see Section 2).

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. This general tool can of course be used for benchmarking other spatail operations. Experimentators can use a standard Web browser to specify the number of rectangles they want in a sample, as well as their distribution with regard to size, shape, and location. Various standard 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 ENST rectangle generator, we defined several test suites and ran experiments to compare the nested loop strategy and two index-based strategies: scan-and-index, and synchronized tree traversal. Our results showed that the relative performance of the two index-based strategies depends on two parameters: selectivity and sample size. As expected, smaller selectivities (i.e., less tuples in the Cartesian product qualifying for the join) clearly favor the index-based strategies. Large selectivities, 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 selectivity, however, the impact of sample size is generally much smaller. The difference between the two index-based strategies was negligible in comparison.

A later implementation will also include an evaluation of other join strastegies. Sort-merge and hash joins are currently being implemented.

We also plan to enrich the rectangle generator so as to include a larger variety of distributions (skewed distributions, correlative x,y distributions, etc.). Our final 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. *Verifiable* means that other researchers should be able to repeat the 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. We invite the reader to access our Web site and do so.

References

[AOT+95] 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.

[BHF93] 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.

[BKS93] 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.

[BKSS90] 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.

[Deu89] O. Deux. The Story of O_2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):91–108, March 1989.

[GR94] 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.

[Gün93] O. Günther. Efficient computation of spatial joins. In *Proc. IEEE 9th Int. Conference on Data Engineering*, 1993.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 47–54, 1984.

[LH92] W. Lu and J. Han. Distance-associated join indices for spatial range search. In *Proc. IEEE 8th Int. Conference on Data Engineering*, 1992.

[LR94] 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.

[LR95] 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.

[LR96] M.-L. Lo and C. V. Ravishankar. Spatial hash-joins. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 247–258, 1996.

[LZ94] 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.

[ME92] P. Mishra and M. H. Eich. Join processing in relational databases. *ACM Computing Surveys*, 24(1):63–113, March 1992.

[Ore86] 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.

[PD96] 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.

[PS85] F. P. Preparata and M. I. Shamos. *Computational geometry*. Springer-Verlag, New York, NY, 1985.

[PTSE95] 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.

[Rot91] D. Rotem. Spatial join indices. In *Proc. IEEE 7th Int. Conference on Data Engineering*, pages 10–18, 1991.

[Sam84] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[SFGM93] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The Sequoia 2000 storage benchmark. In *Proc. ACM SIGMOD Conference on Management of Data*, 1993.

[SGR96] 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.

[TLPH95] 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.

[Val87] P. Valduriez. Join indices. *ACM Trans. on Database Systems*, 12(2), 1987.

Appendix A: Related Web Pages

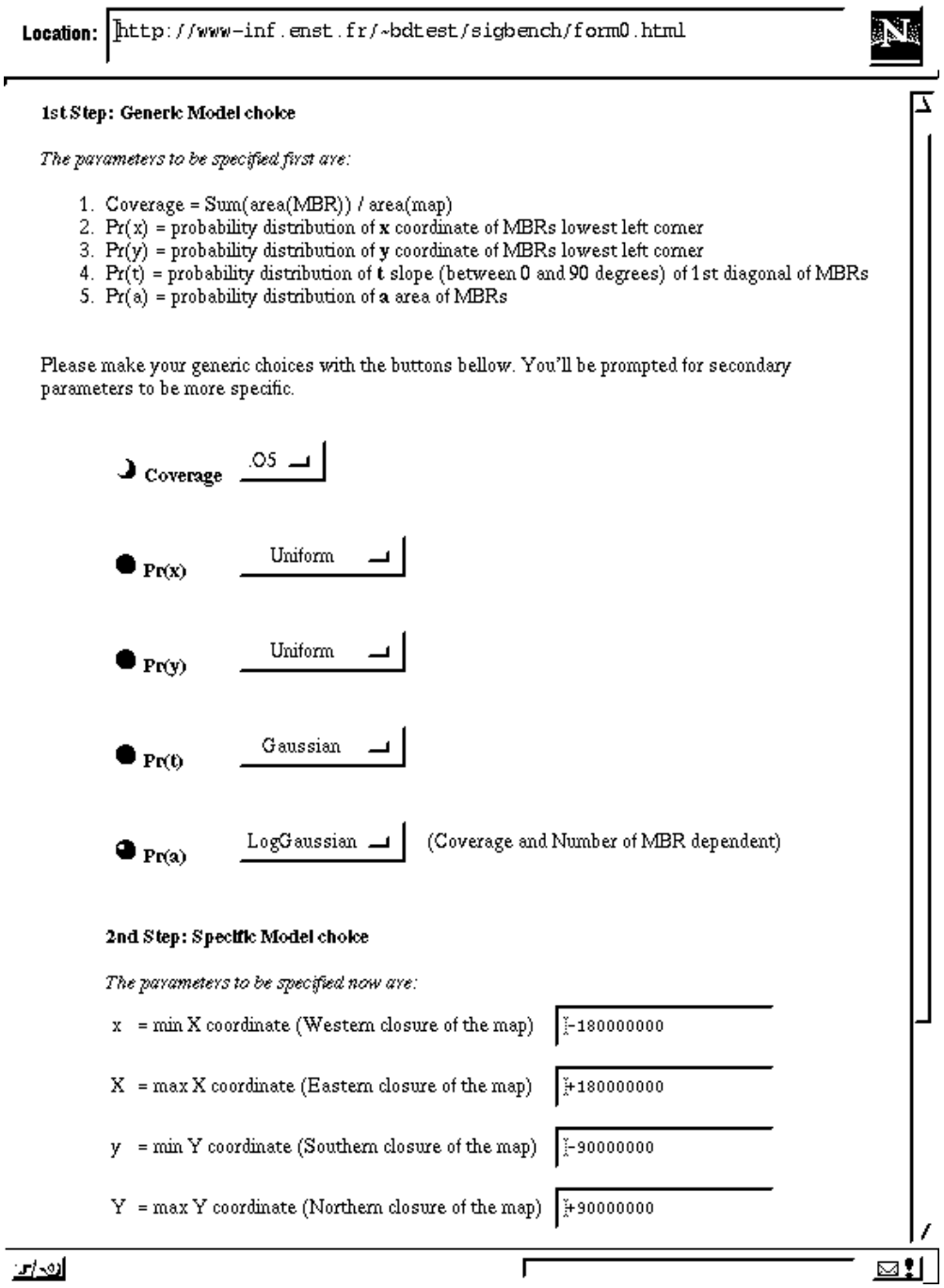

Figure 8: The Web interface to the rectangle generator (page 1)

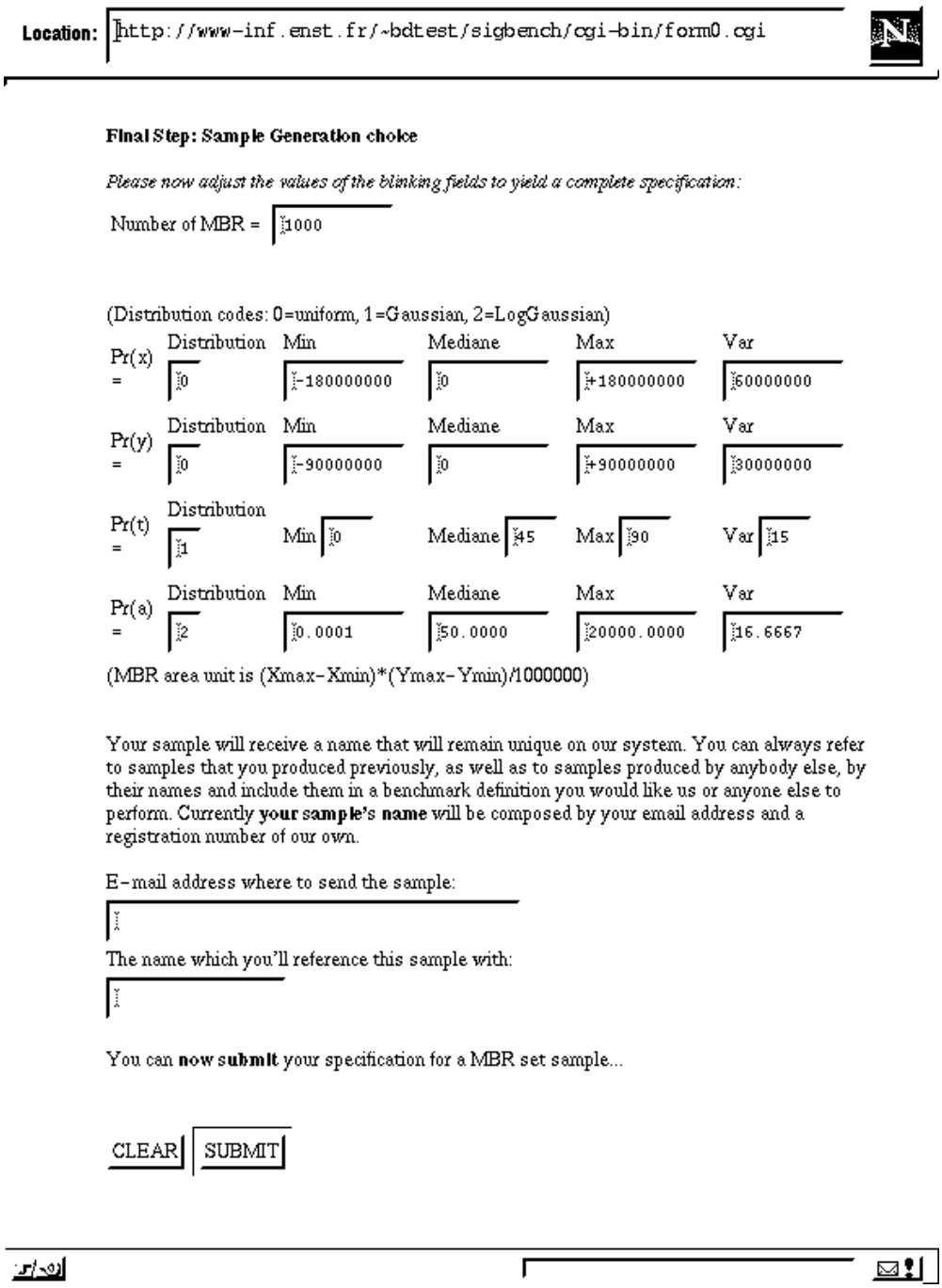

Figure 9: The Web interface to the rectangle generator (page 2)

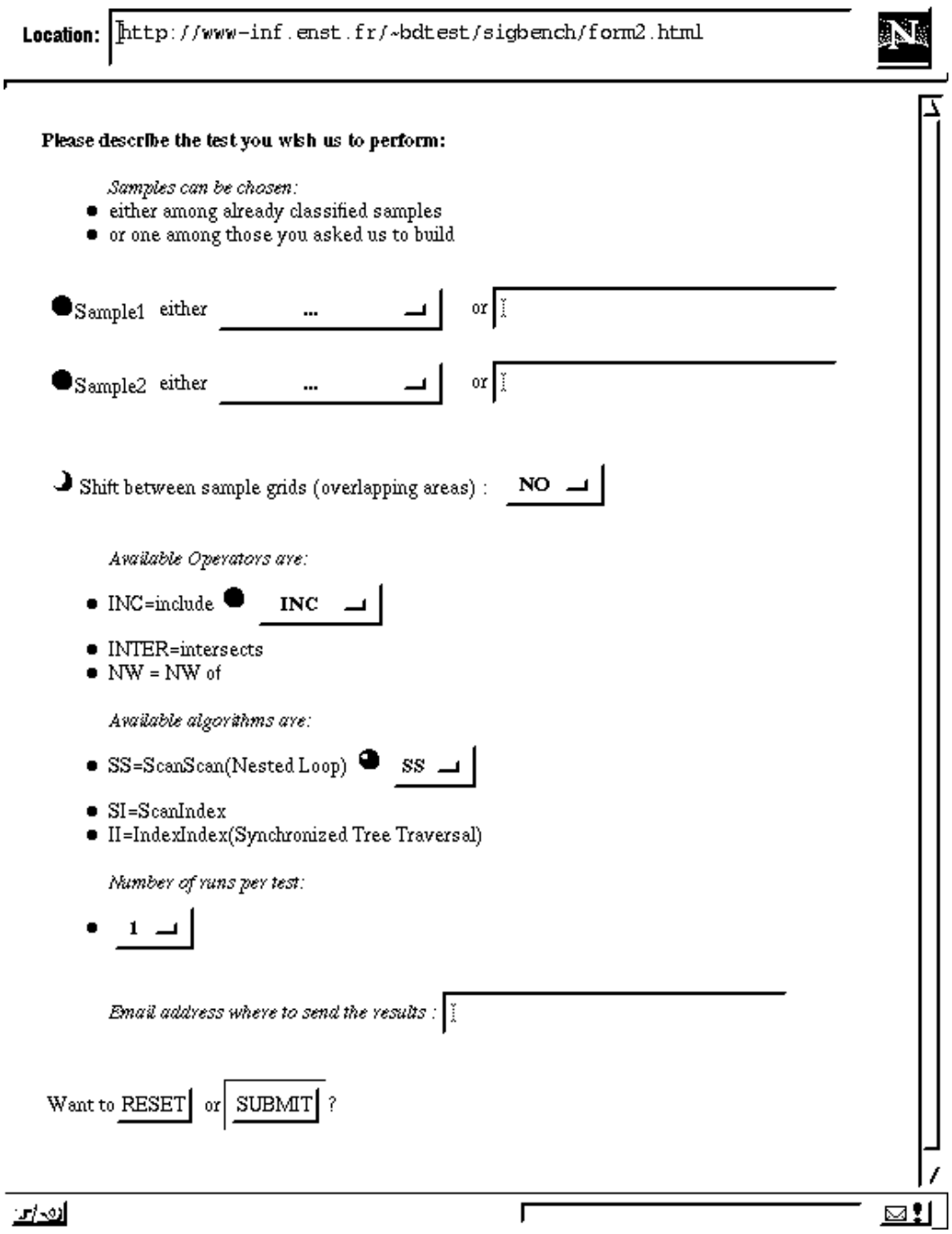

Figure 10: The Web interface to running experiments