À La Carte: A Web-Based Benchmark for Spatial Database Joins*

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

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 the selectivity factor 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 collections of objects that are each associated with a spatial location and a spatial extension. In a relational or object-oriented database system, each

*Earlier versions of parts of this paper were presented at the *13e Journées Bases de Données Avancées* (Grenoble, 1997) and at the *10th Int. Conf. on Scientific and Statistical Database Management* (Capri, 1998).

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

‡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`

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

of these collections would typically be represented by a relation or a class, respectively. *Within 10 kilometers of* and *belong to* are spatial operators that serve as join predicates.

Günther [Gün93] 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 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 join predicate θ, there are many possibilities, including

- *intersects*
- *contains*
- *enclosed_by*
- *distance* λ q, with $\lambda \in \{=, \leq, <, \geq, >\}$ and $q \in \Re_0^+$
- *northwest_of*
- *adjacent_to*

Intersection is perhaps the most important join predicate [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 join predicates is not immediately obvious [BKS93, PD96, LR94, LR95, LR96].

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, 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 [PD96], 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 [SFGM93].

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	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
U	size of the universe: $(x_{max} - x_{min})(y_{max} - y_{min})$

Table 1: Parameters of the rectangle generator

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 [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 benchmark design, the algorithms that are currently available, and various implementation issues. Section 3 presents the results of our experiments. Section 4 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. 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-inf.enst.fr/~bdtest/sigbench/menu.html. Users can transmit parameters to the generator and obtain a corresponding random sample. Figures 13–15 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.

2.2 Workloads

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 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 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 [SFGM93]. Fig. 4 displays one of these samples and shows the skewed distribution of the objects in the plane.

2.3 Algorithms

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

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

Biotopes-N

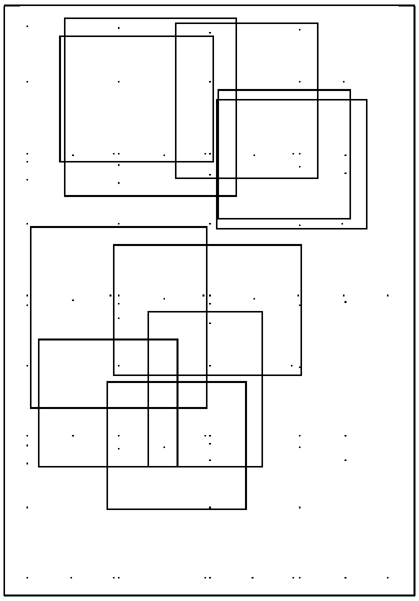


Figure 1: *Biotopes* model, N=10

indices [ME92]. The application of these techniques to spatial joins is not always straightforward.

Sort-merge, for example, often does not work because there is no total ordering among spatial objects that completely preserves spatial proximity [Gün98]. As a result, for many join predicates there is no sort that makes sure that one catches all matching tuples during the following merge. One notable exception from this effect is the join predicate *intersects*, for which sort-merge strategies can be used. Efficient implementations have been described, among others, by Orenstein [Ore86], Abel et al. [AOT+95], and Becker [BHF93].

In this paper we concentrate on three strategies to compute spatial joins: nested loop (NL), scan-and-index (SI), and synchronized tree traversal (STT). We discuss these approaches in turn.

2.3.1 Nested Loop (NL)

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.

2.3.2 Scan-and-Index (SI)

The *scan-and-index* strategy (also called *index-supported joins*) can be applied if at least one of the relations involved has an index that supports the join predicate θ, i.e., the retrieval of matching tuples. Consider, for example, a relation R with a B^+-tree on the relevant column i and θ being a simple comparison predicate ($=, <, \leq, >, \geq$). One may now scan the other relation

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

Cities-N

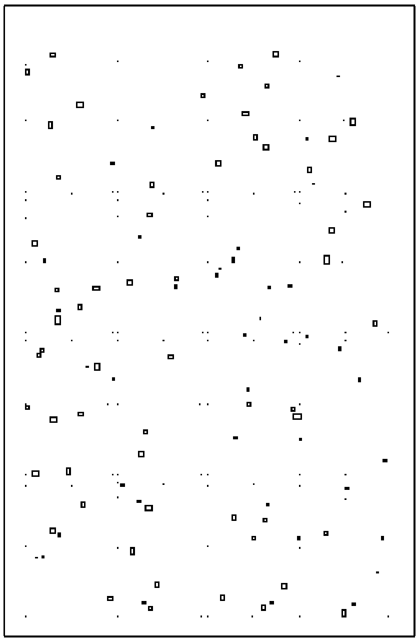


Figure 2: *Cities* model, N=100

(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 both of the relations involved. A typical example would involve the use of an R-tree [Gut84] to obtain spatial objects that intersect or have a certain maximum distance from another object.

2.3.3 Synchronized Tree Traversal (STT)

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*. 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 [Gut84] and the R*-tree [BKSS90]. 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

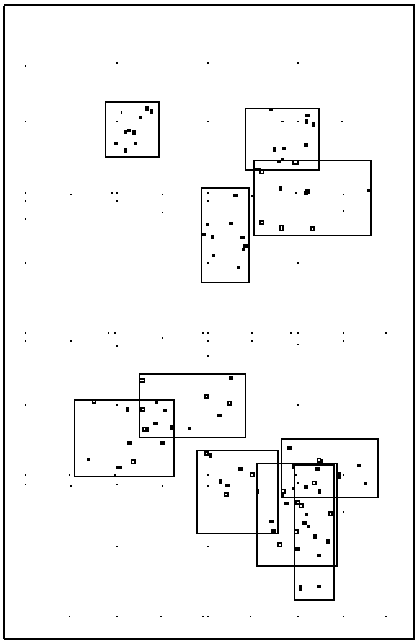
$x_I, y_I \sim \mathcal{U}(0,1)$ $t_I \sim \mathcal{N}(\pi/4, \sigma)$ $a_I \sim \mathcal{N}(0.3/N_I, \sigma)$	
$x_{II}, y_{II} \sim \mathcal{U}(0,1)$ $t_{II} \sim \mathcal{N}(\pi/4, \sigma)$ $a_{II} \sim \mathcal{N}(1/N_{II}, \sigma)$	

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

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 join predicates θ.

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 $root(GT_{R.A})$ and b' of $root(GT_{S.B})$ the condition $a'\Theta b'$ is true. For each qualifying

$o_1\theta o_2$	$o'_1\Theta o'_2$
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: Join predicates θ and corresponding Θ-predicates.

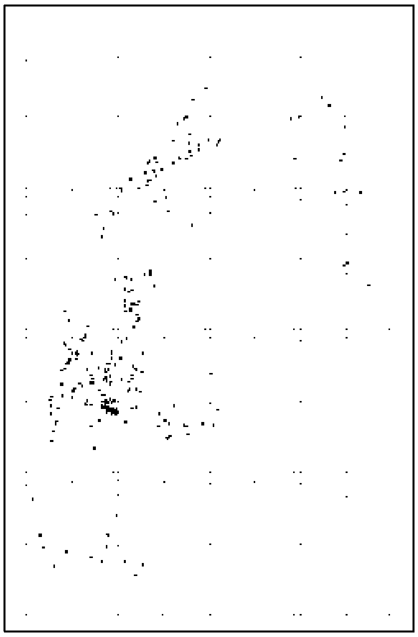


Figure 4: *Sequoia-16*: a real-life model

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 [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, 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. The application of R-trees to predicates other than *intersects* has been discussed by Papadias et al. [PTSE95].

2.4 Implementation Issues

As a testing environment, we chose the object-oriented database system O_2 [Deu90]. All algorithms were implemented under O_2 version 4.5 and SunOS 4.1.3 on a Sun workstation Sparc 10.

For SI and STT, we used an efficient secondary-memory implementation of a special quadtree data structure [SGR96]. 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.

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 a large number of *tests*, each defined by a join predicate and the models underlying the two input relations. Table 3 gives several examples. All tests compare a *Biotope* sample to either another synthetic sample or a sample from the Sequoia 2000 storage benchmark: *(Sequoia-11)* containing 7973 rectangles, and *(Sequoia-16)* with 971 rectangles. The column *sample size* contains the size of the Cartesian product of the two model samples. We consider three sample sizes: *large* (7.7×10^6), *medium* (7.7×10^5), and *small* (7.7×10^4). This seemingly odd choice of sample sizes results from the given size of the *Sequoia* samples. We adapted the size of the synthetic samples accordingly.

We concentrated on two join predicates: *distance* $\leq \lambda$, and *northwest*. Note that this includes the notorious *intersects* predicate because it is equivalent to *distance* $= 0$. Both predicates, *distance* and *northwest*, allow the convenient manipulation of their *selectivity factor*. Selectivity factor is defined as the ratio between the number of tuples qualifying and the number of tuples in the Cartesian product (i.e., the sample size). In the case of *distance*, the selectivity factor monotonically increases towards 1 with increasing λ. Similarly, the selectivity factor of *northwest* can be manipulated by shifting the universe underlying one of the two samples.

Model 1	Model 2	Sample Size
Biotopes-80	Biotopes-971	77680
Biotopes-80	Cities-971	77680
Biotopes-80	Continents-971	77680
Biotopes-80	Sequoia-16	77680
Biotopes-797	Biotopes-971	773887
Biotopes-797	Cities-971	773887
Biotopes-97	Cities-7973	773381
Biotopes-797	Continents-971	773887
Biotopes-97	Continents-7973	773381
Biotopes-797	Sequoia-16	773887
Biotopes-97	Sequoia-11	773381
Biotopes-971	Cities-7973	7741783
Biotopes-971	Continents-7973	7741783
Biotopes-971	Sequoia-11	7741783

Table 3: Tests performed

Each test is performed with each of the three algorithms. To eliminate measuring errors and externalities, most tests were run several times, both with identical and with different random samples. The variations in elapsed time proved consistently negligible.

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. 13–15 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.

3 Experimental Results

3.1 Major Trends

Our three main results are:

1. Except for very high selectivity factors (greater than 0.25) it is always worth to use an index-based technique (SI or STT).
2. The advantage of both SI and STT compared to NL decreases with larger selectivities.
3. SI consistently outperforms STT.

Figures 5–8 prove these statements in a convincing way. Note that all of the following performance figures plot *selectivity factor* on the x-axis and the *relative performances* of SI and STT vs. NT on the y-axis. Relative performance is computed as the ratio between the elapsed time for SI/STT and the elapsed time of NL. Sample size refers to the numbers of tuples in the Cartesian product, |Sample 1| $\times$ |Sample 2|; sample size may be *large* (7.7×10^6), *medium* (7.7×10^5), or *small* (7.7×10^4). On our Sparc 10 workstation, the absolute elapsed times for NL were about 2 minutes for the small samples, 20 minutes for the medium samples, and 200 minutes (!) for the large samples. These times makes clear that the use of suitable index mechanisms is an important issue to consider.

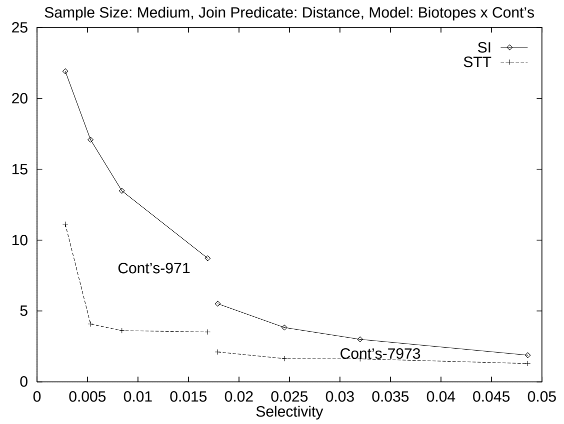


Figure 5: Medium sample, distance

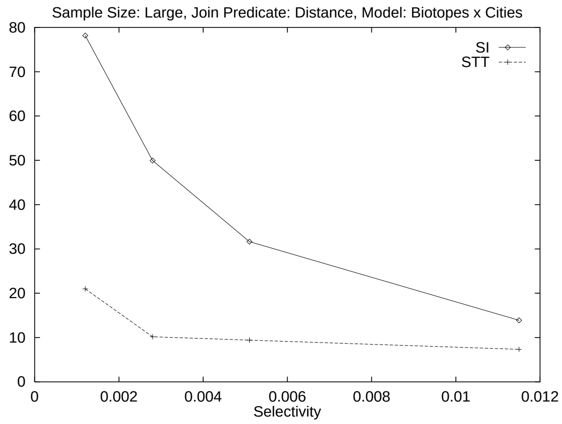


Figure 6: Large sample, distance

As Figures 5–8 show clearly, selectivity factors are an excellent indicator of the observed variations in relative performance. Larger selectivity factors generally tend to lower the performance gains obtained from using an index. For very large selectivity factors, the use of indices is no longer worthwhile because the associated overhead outweighs any potential performance gain. Depending on the particular circustances of the test, the crossover point (gain = 1) occurs

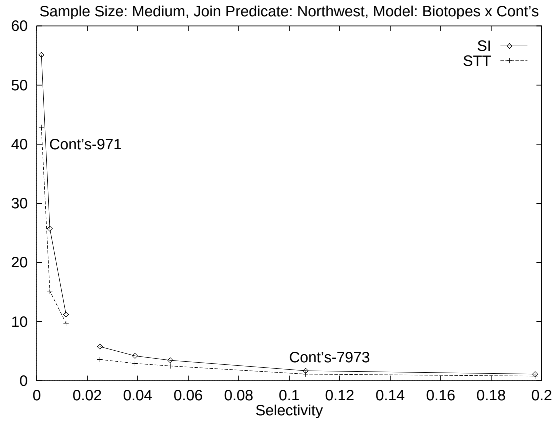


Figure 7: Medium sample, northwest

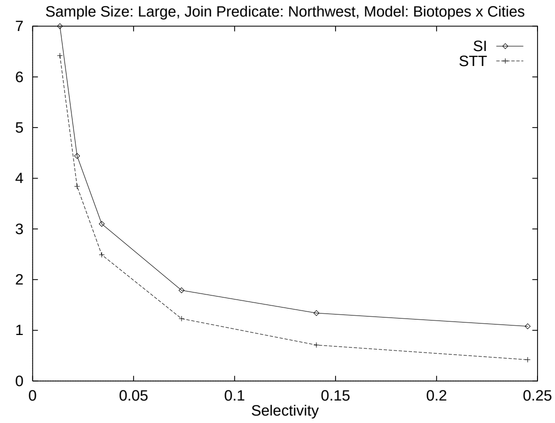


Figure 8: Large sample, northwest

at selectivity factors between 0.2 and 0.3. For smaller selectivity factors, both SI and STT provided significant performance improvements compared to NL. In some cases, performance gains exceeded a factor of 100.

3.2 Minor Trends and Invariants

Other parameters had a lesser impact on the relative performance of the three algorithms. The parameter with the greatest impact after selectivity factor seems to be sample size. Figure 9 shows that larger sample sizes tend to increase the performance advantage of index-based strategies compared to NL.

The choice of join predicate, on the other hand, seems to have no obvious impact on the relative performance. Figure 10 gives no conclusive evidence for significant differences between the two chosen predicates *distance* and *northwest*. The only aspect that counts is the selectivity factor.

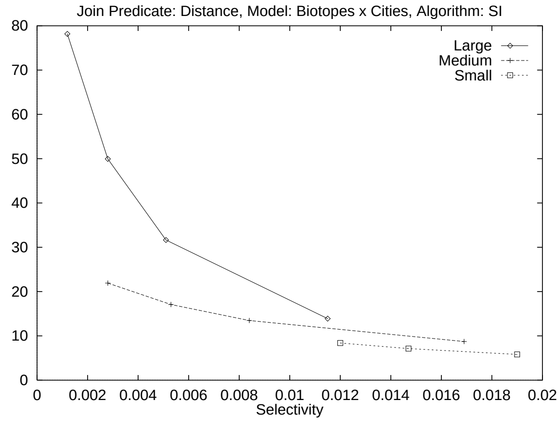


Figure 9: Comparing sample sizes

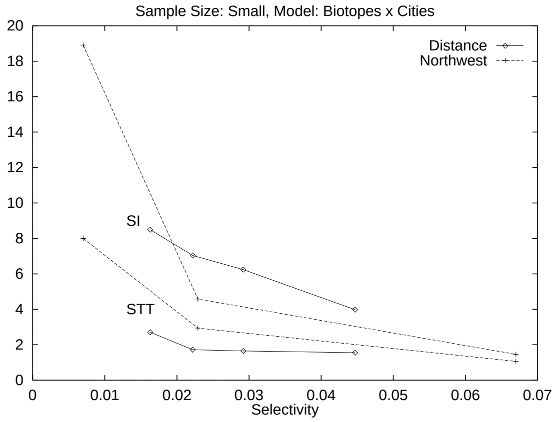


Figure 10: Comparing join predicates

To our surprise, the choice of model (i.e., *Biotopes* vs. *Cities* vs. *Continents*) did not have a significant impact on the measurements either. Figures 11 and 12 plot the behavior of different models for identical selectivities and sample sizes. No clear trend can be extracted.

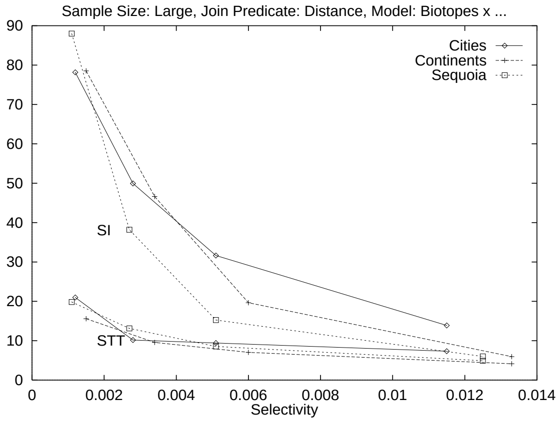


Figure 11: Comparing models (large samples)

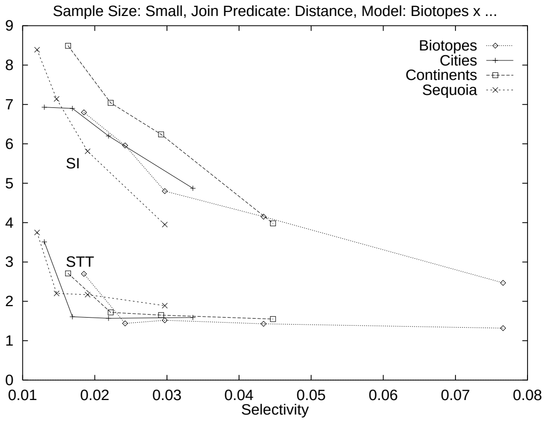


Figure 12: Comparing models (small samples)

4 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 nested loop mainly depends on the selectivity factor of the join predicate. As expected, smaller selectivity factors clearly favor the index-based strategies. Very large selectivity factors, on the other hand, render these strategies virtually worthless because the overhead associated with an index outweighs any potential performance advantages. Other factors like sample size, choice of model, or spatial distribution did not seem to have a major impact on the relative performance of the three algorithms. This should be regarded as a positive result; it means that query optimizers can concentrate on the selectivity factor without getting involved with other specifics of the given data sets.

A later implementation will also include an evaluation of other join strategies, in particular the promising use of join indices [Rot91, LH92, LZ94] and hash joins [LR96]. 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

[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.

[Deu90] O. Deux et al. The story of O_2. *IEEE Trans. on Knowledge and Data Engineering*, 2(1):91–108, March 1990.

[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.

[Gün98] O. Günther. *Environmental information systems.* Springer-Verlag, Berlin/Heidelberg/New York, 1998.

[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.

[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.

[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.

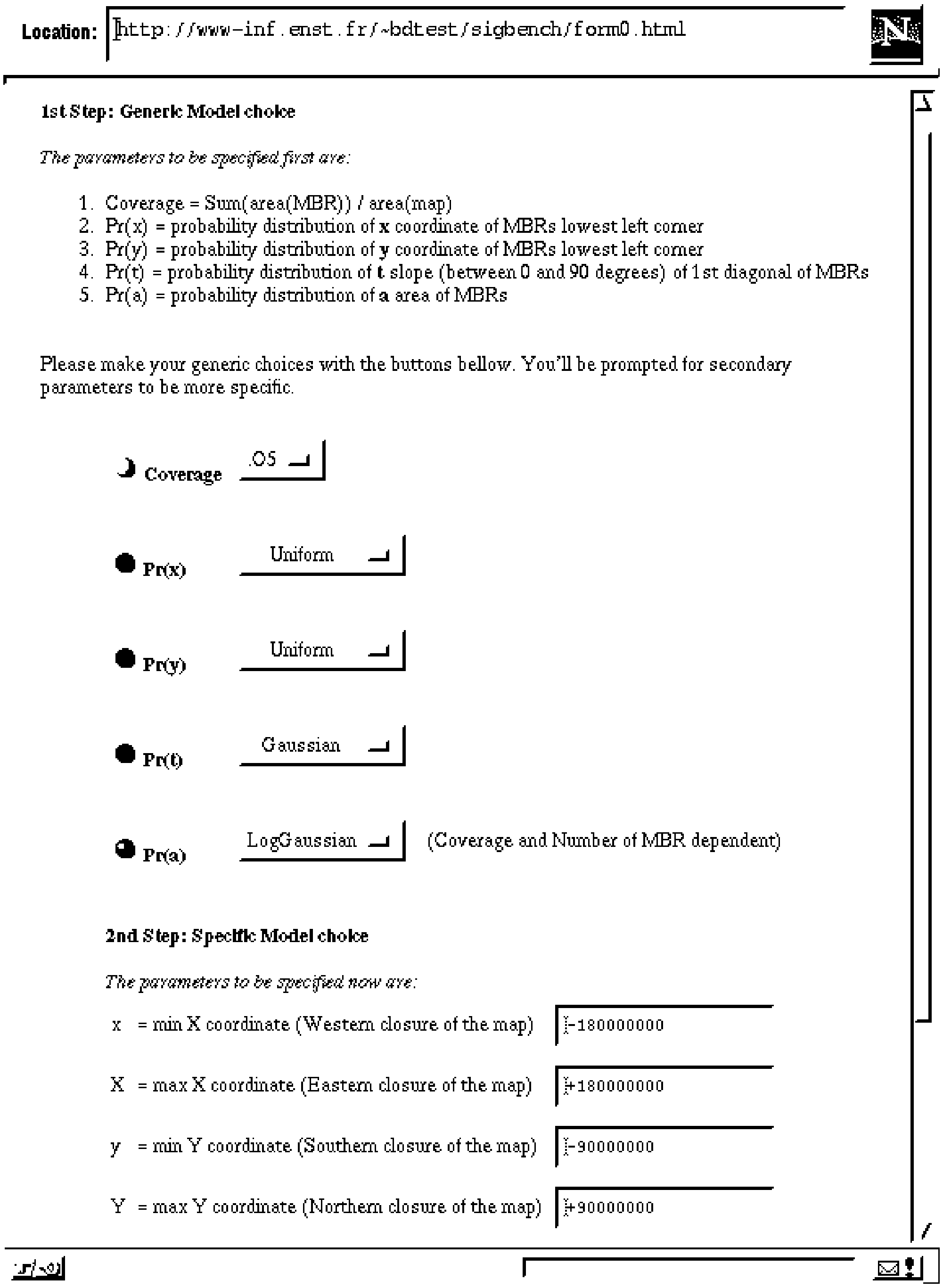
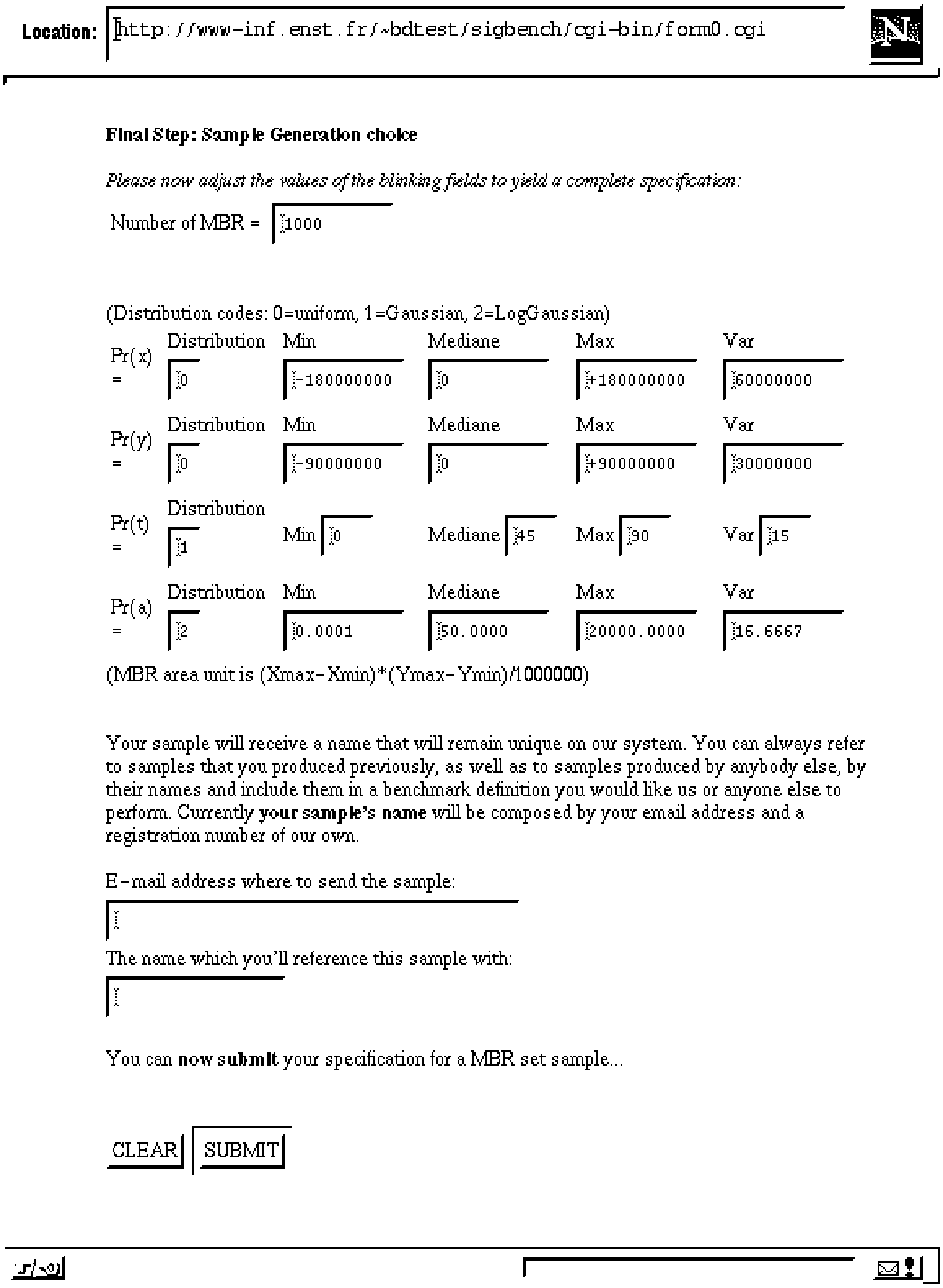


Figure 13: The Web interface to the rectangle generator (page 1)



Location: http://www-inf.enst.fr/~bdtest/sigbench/cgi-bin/form0.cgi

Final Step: Sample Generation choice

Please now adjust the values of the blinking fields to yield a complete specification:

Number of MBR = 1000

(Distribution codes: 0=uniform, 1=Gaussian, 2=LogGaussian)

	Distribution	Min	Mediane	Max	Var
Pr(x) =	0	-180000000	0	+180000000	60000000
Pr(y) =	0	-90000000	0	+90000000	30000000
Pr(t) =	1	0	45	90	15
Pr(a) =	2	0.0001	50.0000	20000.0000	16.6667

(MBR area unit is (Xmax−Xmin)*(Ymax−Ymin)/1000000)

Your sample will receive a name that will remain unique on our system. You can always refer to samples that you produced previously, as well as to samples produced by anybody else, by their names and include them in a benchmark definition you would like us or anyone else to perform. Currently **your sample's name** will be composed by your email address and a registration number of our own.

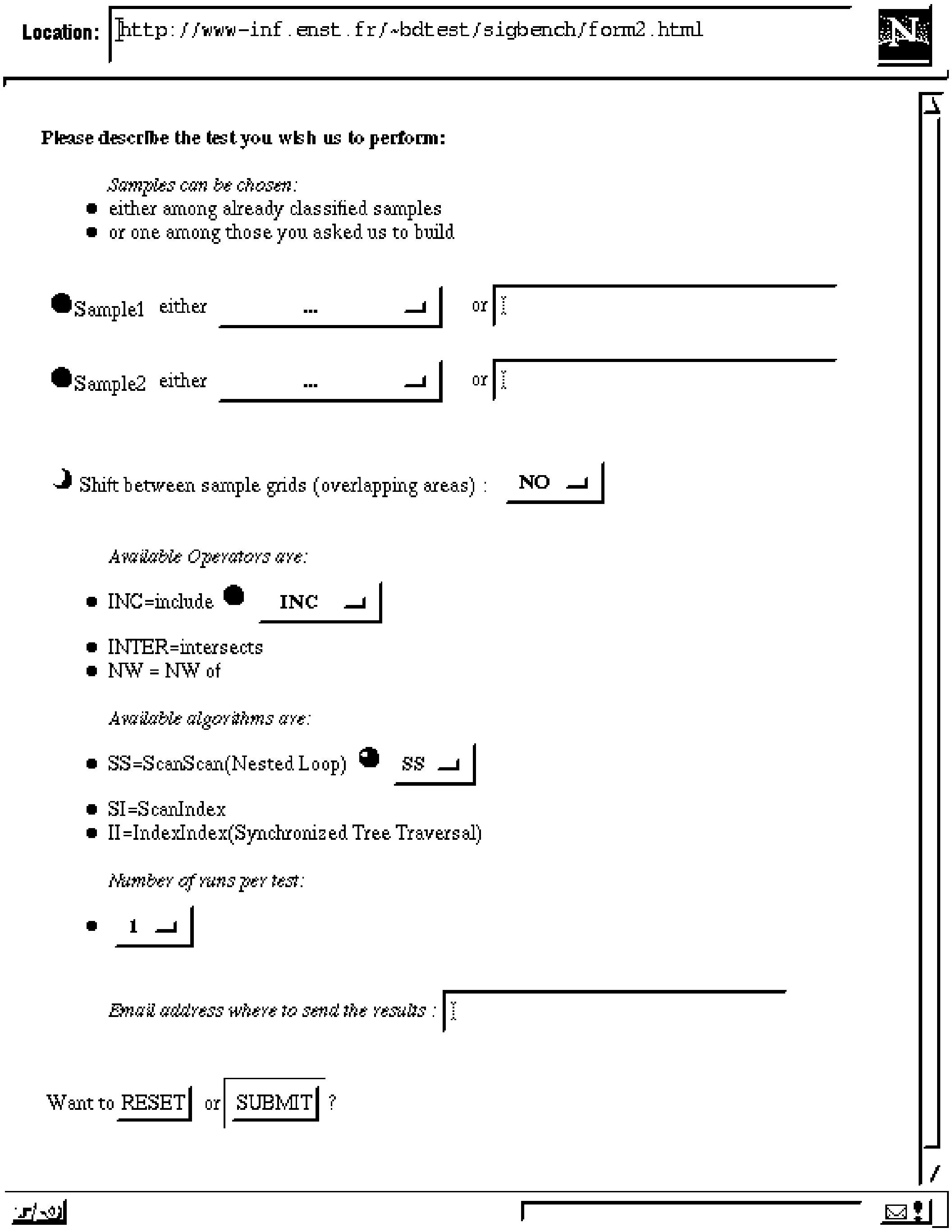
E−mail address where to send the sample:

The name which you'll reference this sample with:

You can **now submit** your specification for a MBR set sample...

CLEAR SUBMIT

Figure 14: The Web interface to the rectangle generator (page 2)



Location: http://www-inf.enst.fr/~bdtest/sigbench/form2.html

Please describe the test you wish us to perform:

Samples can be chosen:

- either among already classified samples
- or one among those you asked us to build

Sample1 either ... or

Sample2 either ... or

Shift between sample grids (overlapping areas) : NO

Available Operators are:

- INC=include INC
- INTER=intersects
- NW = NW of

Available algorithms are:

- SS=ScanScan(Nested Loop) SS
- SI=ScanIndex
- II=IndexIndex(Synchronized Tree Traversal)

Number of runs per test:

- 1

Email address where to send the results :

Want to RESET or SUBMIT ?

Figure 15: The Web interface to running experiments