THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Lower Kent Ridge Road, Singapore 119260

TRA7/07

Non-blocking Spatial Join

Wee Hyong TOK, Stephane BRESSANE and Mong Li LEE

July 2007

Technical Report

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

JAFFAR, Joxan
Dean of School

Non-Blocking Spatial Joins

Wee Hyong Tok Stéphane Bressan Mong Li Lee
School of Computing
National University of Singapore
{tokwh,steph,leeml}@comp.nus.edu.sg

Abstract

We propose and study sequential non-blocking algorithms for the processing of spatial joins on continuous data streams with unpredictable arrival rates or on large collections of spatial data that are not indexed. Given two sets of spatial data represented by their bounding boxes, the algorithms immediately and continuously compute and output the pairs of data from each set whose bounding boxes intersect. The different algorithms we propose take advantage of different possible characteristics of the data such as clustering of the input to build indexes or synopses to accelerate the production of results. We comparatively analyze the performance of the proposed algorithms using several synthetic and realistic data sets.

1. Introduction

Algorithms for the processing of spatial join, such as those proposed in [5, 15, 16, 19, 3], assume that the data is organized and readily available on local disks. These algorithms emphasize the efficient processing of the complete result of the spatial join. We refer to these algorithms as *blocking spatial join* algorithms for they require both data sets to be available and possibly indexed before results are requested.

The modern information infrastructure is one of networked devices possibly mobile and wireless. It enables the production and consumption of huge amounts of data. The applications feeding on these data either need to process continuous streams of spatial data or require the processing of quantities of spatial data so huge that they render the existing blocking algorithms impractical for a user waiting for results. They compel spatial join algorithms that can swiftly deliver initial results with the minimum negative impact on the overall response time, i.e. *non-blocking spatial join* algorithms.

The first family of parallel non-blocking spatial join algorithms is proposed and studied in [17]. While the focus of the work presented in the above mentioned paper is on achieving speed-up by distributing the task of performing the spatial join amongst several processors, the authors considered non-blocking spatial join with a transient in-memory R-tree index structure.

Our work focuses on exploring the possible design space for sequential non-blocking spatial join. The result of our study can be the basis for further studies investigating parallel versions of the algorithms we present and discuss but also offers solutions to applications which can not deploy a parallel computing infrastructure.

The remainder of the paper is organized as follows. In Section 2 we present related work. In Section 3, we discuss those algorithms that constitute the natural non-blocking extension of existing spatial join algorithms. Reflecting on the drawback of the R-tree based non-blocking algorithms, in Section 4, we present a symmetric block nested loop join algorithm. In Section 5, we then present an algorithm that tries and combines the advantages of the algorithms presented before in the favorable situation where data on both streams are sufficiently clustered. In Section 6, we empirically evaluate the proposed techniques using both synthetic and realistic data sets and discuss and analyze the results. Finally, in Section 7, we summarize our results and contribution and outline further possible development of our work.

2. Related Work

Spatial join algorithms leverage existing spatial index structures such as R-tree [8], R+-tree [23], R*-tree [4] and PMR quad-tree [22].

In [21],the authors propose the use of join indices for grid files. In [11], the authors conducted a comprehensive study on the use of R-tree, R+-tree, R*-tree and PMR quad-tree for spatial queries for databases containing large line segments. We follow the algorithms of the family of the spatial join based on R-tree proposed in [5]. In this paper, the authors assume the two R-trees are readily available for the join. The authors of [12] propose a breadth-first strategy

for the join of the two R-trees. The work reported in [18] extends the above results to multi-way spatial joins.

As discussed in [6] spatial join algorithms first filter the data according to their bounding boxes before they can refine the results by computing the data whose actual shapes intersect. We are primarily concerned in this paper with the filter phase and consider the join of two incoming streams of bounding boxes. We assume that the refining phase can be applied to the filtered data using any of the already proposed in-memory techniques. Such in-memory techniques have been introduced in [19] and in [3] in which partitioning of the data space in chunks and stripes respectively is used to filter data before the results are refined using computational geometry algorithms.

In [15], the authors consider the case in which only one of the two data sets to be joined is already indexed with an R-tree. The method, called the seeded tree method, uses the top k level of the R-tree of the indexed data set to seed the construction of an R-tree for the second data set. The same authors [16] propose a spatial hash join. The two data sets are respectively partitioned into spatial buckets using the top k levels of an R-tree. Data is replicated into every bucket it overlaps before data of the two respective data sets can be joined within each bucket.

In the relational context, several non-blocking query processing operators are proposed, e.g. [24, 25, 10, 13]. Query scrambling techniques [25] allow the modification of query execution plans at run-time in order to adapt to the delays from the data feeds. In [14], the authors further introduce a mid-query *reoptimization* scheme to allow the query execution plan to be modified based on the size of the intermediate result. The CONTROL project [10] proposes a model for pipelining query processing algorithms in order to provide sufficient feedback to influence the runtime behavior. In [9], a family of *Ripple Join* algorithms is introduced where the statistics gathered at runtime was used are used to tune the join's behavior. XJoin [24] extended the symmetric hash join [2] to allow both in-memory data and disk partitions. Inter-arrival delays are taken advantage of to perform joins for partial partitions that have been spooled to secondary storage. This helps to improve the overall throughput of the join as well as the early production of data.

Following the work on parallel spatial join by [7], the authors of [17] propose a parallel non-blocking spatial join algorithm, which uses in-memory R-trees at each node. The R-trees are built and freed whenever data is re-distributed.

3 R-tree Based Blocking and Non-Blocking Spatial Joins

3.1 Static Spatial Join

Let us first recall the general strategy of an R-tree based blocking spatial join. For the sake of using the performance of this algorithm as a base line in the subsequent performance analysis, we can consider, without loss of generality for the non-blocking algorithms that we propose, that the data sets are bounded. The blocking R-tree based spatial join first builds two R-trees one for each incoming data set. We do not use bulk loading which would further delay the production of results. When all the data has arrived, a synchronized traversal of the R-tree is used to compute overlapping data (since we are concerned only with the filtering phase, data consist of an identifier to refer to the actual spatial object and the four coordinates of a minimum bounding rectangle.) This strategy is the basis of algorithms such as those in [5, 15, 16, 19, 3] even though details of the underlying data structure and algorithms might differ. Since we consistently use R-trees, we believe that the relative performance is generally similar to the one we would obtain with more sophisticated index structures such as R+-trees, R*-trees, and their variants.

Figure Algorithm 1 outlines the algorithm for joining the two data sets *R* and *S* by constructing the two R-trees P_R and P_S to guide join processing.

Algorithm 1: Static Spatial Join Algorithm

```
Data   : Spatial Relations R and S
Result : MBRs that overlaps from R and S
begin
    /* Build Phase */
    /* P_R and P_S are intermediate data structures */
    for Tuple t ∈ R do
        Insert t into P_R
    end
    for Tuple s ∈ S do
        Insert s into P_S
    end
    /* Join Phase */
    Join(P_R,P_S)
end
```

We can identify two distinct phases in this generic framework. In the *Build* phase the index is build. In the *Join* phase the indices are use to guide the production of results. If extending this algorithm, non-blocking algorithms need to interleave the build and join phases in order to allow the early production of results.

3.2 Fully Dynamic Spatial Join

The first non-blocking algorithm that comes to mind, considering the above discussed blocking algorithm, consists in the interleaving of the two phases at finest granularity. A system-based concurrent execution of the two phases that would rely on concurrency control of the R-tree accesses is not necessary since each insertion preemptively locks the root of the tree to allow potential splits to retro-propagate up to the root if necessary. It suffices to programmatically alternate the two phases. Namely each incoming data from either set is inserted into its corresponding R-tree and used to probe the other data sets already partially build R-tree. We call this algorithm the *fully dynamic spatial join*.The fully dynamic spatial join algorithm is outlined on Algorithm 2.

Algorithm 2: Fully Dynamic Spatial Join Algorithm

```
Data   : Spatial Relations R and S
Result : MBRs that overlaps from R and S
begin
    while (Data Available) do
        Read a tuple from either of the data source_i ;
        Insert tuple into R-tree, R_i ;
        Probe other R-tree using tuple ;
        Return matches if found ;
    end
end
```

Clearly, this algorithm will produce the first results very early. Yet we can expect its overall performance to be much worse than the one of the blocking spatial join algorithm. This is noted in [15]: *If we simply used R-tree and let them overflow to disk when they grow larger than main memory, performance would not be acceptable.*. Indeed one of the dominant costs, namely the amount of retrieval of data pages from disk, is commensurate to the amount of probing (ultimately the sum of the size of both data sets.) Although this cost is reduced by the use of a buffer and by an adequate replacement policy such as the *least recently used* or *LRU* policy, it can only be done within the limit of the available space available for the buffer (a fortiori so if data sets are unbound).

3.3 Block Fully Dynamic Spatial Join

In order to seek a compromise between the minimum number of input-output operations required by the blocking algorithm and the non-blocking behavior of the fully dynamic algorithm, we propose to alternate the insert and join phases for blocks of data. Namely whenever we have received a predefined number of data to form a block from one of either set, the block of data is inserted into its corresponding R-tree and the disjunctive list of data is used to probe the other data sets already partially build R-tree. We call this algorithm the *block fully dynamic spatial join* algorithm.

The algorithm is outline on Algorithm 3.

Algorithm 3: Block Fully Dynamic Spatial Join Algorithm

```
Data   : Spatial Relations R and S, BlkThreshold T
Result : MBRs that overlaps from R and S
begin
    while (Data Available) do
        Receive a tuple_i from either of the data source_i;
        Insert tuple_i into TupleCollection_i;
        if (size(TupleCollection_i) >= T) then
            Insert all tuples in C_i into R-tree, R_i ;
            Use all tuples in C_i to probe the corresponding R-tree;
            Return matches if found ;
            Empty TupleCollection_i;
        end
    end
end
```

The size of the blocks determines the compromise between the early production of results (small blocks) and the overall performance (large blocks). For reasons of symmetry (assuming identical arrival frequency on both data sets) a size of half of the buffer yields the optimum overall performance.

3.4 R-tree Based Non-Blocking Spatial Joins

The R-tree based non-blocking spatial joins should yield interesting performance in the production of early results while not compromising the performance of the overall production of results as long as the input-output time saved by retrieving relevant pages thanks to the R-tree and the cpu time saved by comparing spatially related data overcomes the cost of the repeated join phase. The questions are whether this cross-over occurs after a sufficient percentage of the data has been produced and how much overhead is incurred at completion of the join (for finite data sets).

In other attempts, whose full details are not reported here, we have considered variants of the non-blocking algorithms described above in which the partially build R-trees are joined instead of being probed with a list of data as well as strategies for inserting data as they arrive and for marking them to avoid duplicate results. The empirical analysis

showed poor performance compared to the algorithms discussed in this paper.

4. Symmetric Block Nested Loop Algorithm

The main purpose of the R-tree is to adaptively create a balanced partition of the data. Other partitioning technique such as grids or quad-trees either degenerate if the data is skewed in a way not captured by the partition or introduce may introduce similar overhead to the one of the R-tree for a similar granularity of partitioning. Given the expected prohibitive cost of managing a disk resident R-tree, we can consider an even more radical solution, namely an algorithm that solely focuses on reducing the input-output operations with respect to the buffer without attempting to partition the data. In conventional relational database management systems, if no relevant index data structure exists on either of the data sets to be joined, one of the most common join algorithms is the Block Nested Loop Join [20]. In this section we propose a *symmetric block nested loop* algorithm. As a matter of fact such an algorithm applies equally to spatial and non-spatial data since no particular organization of the data is needed which depends on its spatial nature. In the relational context with adequate join conditions on pairs of attributes one can consider efficient dynamic partitioning functions such as hash functions (yielding algorithms such as the Xjoin [24], for instance). Such partitioning functions so far have found no equivalent in the spatial domain, and are not readily available for arbitrary join conditions in general in other domains.

4.1 Algorithm

The fact that we are dealing with pages instead of data elements allows us a tighter control of the buffer. In the symmetric block nested loop algorithm we partition the buffer of size B into three groups. We allocate two buffers of $n = (B-1)/2$ frames (to hold one block of n pages) to read in data from each of the two data sets. Two counters are kept to indicate when a full block of data is read from either data set. When full, the block of data is joined in a nested loop with the already disk resident data of the other data set. In addition, a single buffer frame is reserved to read from the disk the data to be joined. The build phase is reduced to reading and storing the data since no index is built. The pages in the each block are written to disk as new data is read according to the LRU replacement policy. This occurs after the data in the buffer have been joined thus not necessitating duplicate elimination in the results. Algorithm 4 outlines this algorithm.

An additional noticeable advantage of this algorithm is that data is written in pages in its order of arrival as opposed to being reorganized as in the algorithms given in the previous section. Provided pages or data are time-stamped, this feature simplifies the task of discarding outdated data if the application requires it.

If the data displays no particular pattern of arrival with respect to its spatial distribution and in the impossibility to find a satisfactory and economic partitioning mechanism, we expect the symmetric block nested loop algorithm to be competitive.

Algorithm 4: Symmetric Block Nested Loop Algorithm

```
Data   : Spatial Relations R and S, BlkThreshold T
Result : MBRs that overlaps from R and S
begin
  while (Data Available) do
    Read a tuple from either of the data source_i;
    Insert tuple into buffer B_i;
    BlkCounter_i++;
    if (BlkCounter_i >= T) then
      For each stored page of the other data set
      Join this page with the data in B_i;
      BlkCounter_i = 0;
    end
  end
end
```

5 Back to the R-tree

We now consider an algorithm suitable for those applications in which data is expected to arrive in spatial clusters. In such a case, except at the transition between two arriving clusters, we can expect a sequence of incoming data from one data set, say of the size of one page, to be spatially near.

5.1 Summary R-tree

Based on the above assumption the algorithm we propose uses an R-tree to index pages instead of individual data (notice that the approach naturally extend to considering groups of several pages if the data sets are very large and the clustering sufficient). For each page of data read from each data set, the minimum bounding rectangle in closing the data in the page is stored in the R-tree for this data set. We call such R-trees *summary R-trees*. Notice that this is different from bulk loading the actual data in the page since we do not index the actual data but the page that contains them. The size of the summary R-tree is much smaller than the one of an R-tree indexing the actual data. Figures 1 and 2 illustrate the layout of the data in the directory and leaf pages in a complete R-tree and in a corresponding summary R-tree, respectively, for the R100C5 dataset (see section 6),

which contains 100K of data and has five clusters. We see that the summary R-tree still contains the five clusters although it is one level shorter.

5.2 Symmetric Indexed Block Nested Loop

This strategy suggests a new algorithm we call the *symmetric indexed block nested loop*. The Symmetric block nested loop follows the block fully dynamic join algorithm of section 3.3. The fact that we are dealing with pages instead of data elements, as in the case of the block nested loop algorithm, allows us a tighter control of the buffer.

In the symmetric indexed block nested loop algorithm we partition the buffer of size B used in the above algorithm into five groups. We allocate three frames to each R-tree. We allocate two buffers of $n = (B-7)/2$ frames to read in data from each of the two data sets. Two counters are kept to indicate when a block of full pages of data is read from either data set. The size of the block is n.

When a block of full pages of data is read, the minimum bounding rectangles of each page in the block is inserted into the corresponding R-tree. The disjunctive list of minimum bounding boxes is used to probe the other already partially build R-tree. The data in the pages retrieved are joined with the data in the pages in the block.

The pages in each block are written to disk as new data is read according to the LRU replacement policy. This occurs after the data in the buffer have been joined thus not necessitating duplicate elimination in the results. The algorithm is presented in Algorithm 5.

Reflecting the natural clustering of the data, the summary R-tree reduces the number of pairs of pages to be selected for joining the data they contain. The question is whether and at which level of clustering this savings overcome the cost of creating and maintaining the summary R-tree.

This algorithm also maintains the advantage that data is written in pages in its order of arrival as opposed to being reorganized as in the algorithms given in the previous section. Provided pages or data are time-stamped, this feature simplifies the task of discarding outdated data if the application requires it although entries in the summary R-tree might need to be discarded or might become obsolete.

Algorithm 5: Symmetric Indexed Block Nested Loop Algorithm

```
Data   : Spatial Relations R and S, BlkThreshold T
Result : MBRs that overlaps from R and S
begin
  while (Data Available) do
    Receive a tuple_i from either of the data source_i;
    Insert tuple_i into bucket B_i;
    BlkCounter_i++;
    if (BlkCounter_i >= T) then
      MBRList = List of Covering MBRs ;
      FoundList = Use MBRList as query windows in the summary R-tree of the other data source ;
      Perform Block-Nested Loop Spatial Join B_i with with pages in FoundList ;
      BlkCounter_i = 0;
    end
  end
end
```

6 Performance Analysis

6.1 Experimental Set-up

The algorithms are implemented in C. The experiments run on a Pentium 4 1.6GHz PC with 512MB RAM under Windows XP Professional. The input-output operations simulate a state of the art disk spinning at 7200rpms yielding an input-output cost of 8ms. We use a 128 frames buffer for all the algorithms. One frame holds one page. The size of a page is 4096 bytes.

We use both synthetic and real-life datasets. Without loss of generality, we do consider data sets that contain an identifier to the actual spatial object as well as its minimum bounding rectangle. One data record is of size 20 bytes. Unless stated otherwise, inter-arrival rate is constant.

The synthetic data sets are generated using a generator similar to the one described in [15]. The generation allows us to control the number of clusters of the original data distribution as well as the selectivity of the join. Two datasets of size N are generated as follows: We first randomly generate C clusters centers for the first data set. For each cluster center, we generate cluster rectangles, CR. Both the length and width of each cluster rectangle is set at 0.2 in our experiments. We assigned $\lfloor N/C \rfloor$ data rectangles to each cluster. The remaining data rectangles are then randomly assigned to any cluster. To control the selectivity S, the second data set clusters are constructed such that $S\%$ of their area overlaps with a cluster from the other dataset. Data from the same cluster are contiguous. For some experiments, when indicated, the data is randomly reshuffled.

The realistic data sets are the Greek roads and rivers [1] and the German roads and railroad lines [1]. A summary of characteristics of these data sets is presented in Table 1.

Datasets	Number of MBRs	# of Clusters
Real-life		
Greece		
Rivers	24,650	-
Roads	23,268	-
Germany		
Railroad Lines	36,334	-
Roads	30,674	-
Synthetic		
R50KC5, S50KC5	50,000	5
R100KC5, S100KC5	100,000	5
R100KC10, S100KC10	100,000	10
R100KC20, S100KC20	100,000	20
R200KC5, S200KC5	200,000	5
R400KC5, S400KC5	400,000	5

Table 1. Datasets Used

6.2 R-tree Based Blocking and Non-Blocking Spatial Joins

In this experiment, we analyze the performance of the three R-tree based algorithms.

We first use two synthetic data sets of 100K each, with 5 clusters each, and with a join selectivity of 25%. We compare the performance of the static spatial join, the fully dynamic spatial join, and the block fully dynamic spatial join.

Figures 3(a), 3(b) and 3(c) report the cumulated number of input-output operations of each of the three algorithms, respectively, at varying percentage of results produced. On the figure the input-output operations occurring during the build phase (insertion of the data and creation of the R-tree) are in grey, while the input-output operations occurring in the join phase are in black. Notice that the figures have different scales on the y-axis.

Figure 3(d) reports the cumulated response time of each of the three algorithms at varying percentage of results produced.

The response time charts confirms that both the fully dynamic and the block fully dynamic joins can produce results early. At what cost for their overall performance?

By design, the build phase of the static spatial join occurs before any results can be produced. Both the fully dynamic spatial join and the block fully dynamic spatial join successfully distribute the build phase and its input-output operations during the incremental production of results. For the fully dynamic spatial join, the input-output cost of joining each individual data is prohibitive. For the block dynamic join the input-output cost remains similar to the one of the static spatial join.

This respectable performance input-output of the block fully dynamic cannot be maintained for the overall response time. Both dynamic algorithms incur a prohibitive cpu cost. Indeed, while the static algorithm is joining the two R-trees in a single depth-first traversal (see [5], for instance), both dynamic algorithms probe the R-trees for each individual or list of minimum bounding rectangles. Their overall performance in response time is worse than the one of the static spatial join. The fully dynamic algorithm can produce more than 20% of the results faster than the static algorithm on this data set. The block fully dynamic algorithm can produce more than 60% of the results faster than the static algorithm on this data set.

6.3 Symmetric Block-Nested Loop

In this experiment, we compare the performance of the symmetric block nested loop (SBNL) algorithm with those of the static spatial join algorithm. We use two synthetic data sets of 100K each, with 5 clusters each, and with a join selectivity of 25%.

Figure 4(a) reports the cumulated number of pages actually compared during the execution of each of the three algorithms, respectively, at varying percentage of results produced. Figure 4(b) reports the cumulated number of input-output operations of each of the three algorithms, respectively, at varying percentage of results produced. Figure 4(c) reports the cumulated response time of each of the three algorithms, respectively, at varying percentage of results produced.

We see that although many more pages are compared by the block nested loop (each pair of pages, one from each data set, is ultimately compared by this algorithm), this is translated in a reasonably low number of input-output operations thanks to the absence of the index data structure to build and probe and thanks to the buffer. Not only the block nested loop can create early results faster than the static algorithm, but it creates all the results significantly faster than the static algorithm. We consistently observed this pattern of performance for all the data sets we have tried (see subsection 6.5).

6.4 Symmetric Indexed Block Nested Loop and Clustered Arrivals

In this experiment, we analyze the effect of clustered arrivals on the symmetric block nested loop and the symmetric indexed block nested loop (SIBNL) algorithms. We use two synthetic data sets of 100K each, with 5 clusters each, and with a join selectivity of 25%. In a first series of measurements the data is clustered as generated, while in a second series of measurements the data from both data sets is randomly reshuffled.

We compare the performance of the static spatial join, the symmetric block nested loop, and the symmetric in-

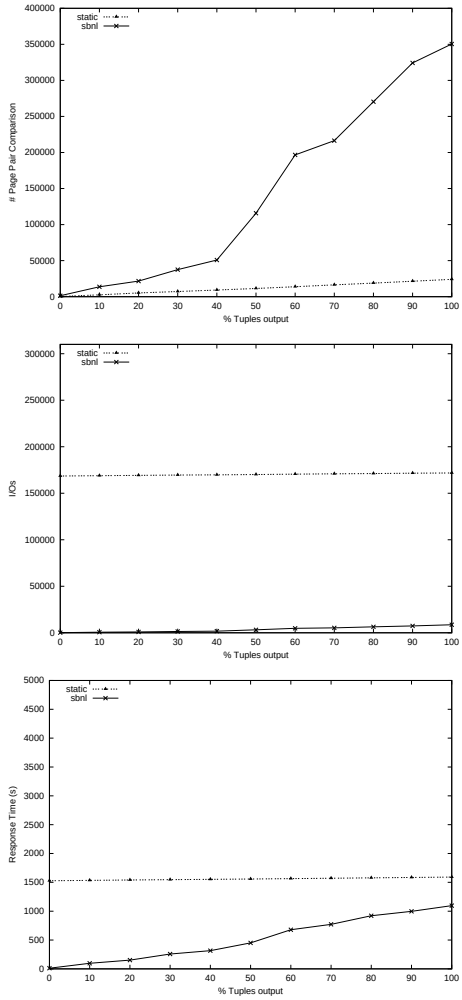


Figure 4. Comparison of spatial joins (Clustered data) (R100KC5 ⋈ S100KC5)

dexed block nested loop for both pairs of data sets.

For the clustered data set, the results are reported on figures 5(a), 5(b), and 5(c) as mentioned in the previous subsection.

We first observed that, as motivated by the design of the symmetric indexed block nested loop, it can reduce the number of pages being compared. This means that it does filter relevant pages of data. Yet because of the cost of maintaining and probing the index data structure, although just a summary, this performance does not translate into a commensurate gain in input-output cost and response time. Nevertheless, with these data sets arriving in clusters, the symmetric indexed block nested loop manages to yield a better response time than the symmetric block nested loop.

For the randomly shuffled data set, figure 5(d) reports the cumulated number of pages actually compared during the execution of each of the three algorithms, respectively, at varying percentage of results produced. Figure 5(e) reports the cumulated number of input-output operations of each of the three algorithms, respectively, at varying percentage of results produced. Figure 5(f) reports the cumulated response time of each of the three algorithms, respectively, at varying percentage of results produced.

The relative performance of the symmetric indexed block nested loop and the symmetric block nested loop as shown on figures 5(b), and 5(c) is now reversed on figures 5(e), and 5(f). This illustrates that the symmetric indexed block nested loop can exploit situations in which data arrive in clusters yet it still performs reasonably well when this is not the case.

The results are accentuated when the data distribution is more clustered and the arrival clustered for instance with 10 and 20 clusters (using R100KC10 ⋈ S100KC10 and R100KC20 ⋈ S100KC20).

6.5 Scalability and Real-Life Data Sets

In this series of experiments, we measure the performance in both input-output operations and response time for the symmetric block nested loop and the indexed block nested loop algorithms on both very large and real-life data sets, respectively. For reference, we also measure the performance of the static spatial join.

We use two groups of synthetic data sets of 200K and 400K. All data sets have 5 clusters and the selectivity is 25%.

For the two input data sets of size 200K, figure 6(a) reports the cumulated number of input-output operations of each of the three algorithms, respectively, at varying percentage of results produced. Figure 6(b) reports the cumulated response time of each of the three algorithms at varying percentage of results produced.

We use two very large synthetic data sets of 400K,

For the two input data sets of size 400K, figure 6(c) reports the cumulated number of input-output operations of each of the three algorithms, respectively, at varying percentage of results produced. Figure 6(d) reports the cumulated response time of each of the three algorithms at varying percentage of results produced.

We use the Greek road and rivers and the German roads and railroads data sets. Figure 7(a) reports the cumulated number of input-output operations of each of the three algorithms, respectively, at varying percentage of results produced for the Greek data. Figure 7(b) reports the cumulated response time of each of the three algorithms at varying percentage of results produced for the Greece data. Figure 7(c) reports the cumulated number of input-output operations of each of the three algorithms, respectively, at varying percentage of results produced for the German data. Figure

7(d) reports the cumulated response time of each of the three algorithms at varying percentage of results produced for the Germany data.

These charts call no further analysis since they are only presented to confirm the analysis in the previous subsections with more challenging data sets.

6.6 Inter-arrival Rate

In this experiment, we illustrate the effect of non-constant inter-arrival times on the various algorithms. We use two synthetic data sets of 50K each, with 20 clusters each, and with a join selectivity of 25%. The inter-arrival is modeled using a Poisson law with a mean of 2 seconds. We compare the performance of the static spatial join, block dynamic spatial join, the symmetric block nested loop, and the symmetric indexed block nested loop.

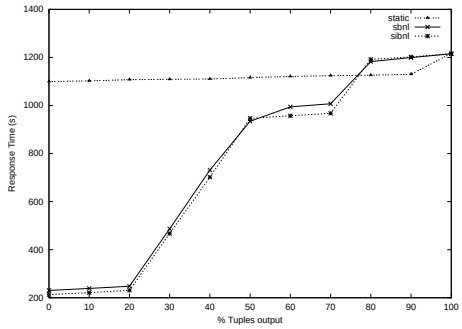


Figure 8. Poisson Inter-arrival with Means at 2s (R50KC5 $\bowtie$ S50KC5)

Figure 8 reports the cumulated response time of each of the four algorithms at varying percentage of results produced for the Poisson law. We can observe that the symmetric block nested loop and symmetric indexed block nested loop were able to produce the initial 75% of the results quickly in spite of the inter-arrival rate.

7. Conclusion and Future Work

In this paper, we have explored the design space for non-blocking spatial join algorithms. We have studied algorithms attempting to use R-trees to dynamically and gracefully partition the data. Yet it appears that the cost of building and probing the R-trees is prohibitive. A simple symmetric block nested loop spatial join, not using any ancillary data structure to partition the data, is the most efficient algorithm.

We have shown however that, when data is known to arrive in clusters, the performance can be further improved by using a summary R-tree which indexes pages of data instead of individual data elements thus reducing the size of the index data structure and the cost of its creation without compromising the partitioning.

As continuation of this work, we are studying the opportunity to build efficient and effective partition function based on a sample of the data. This would allow for instance to efficiently process data that do not arrive in clusters but in chunks that are representative samples of the overall distribution (for instance periodic data as could be produced by sensors). We have already started experimenting with the creation of R-trees from representative samples of larger data sets.

In general, we believe that non-blocking spatial join algorithms can only outperform the symmetric block nested loop when additional knowledge about the data sets characteristics can be assumed and exploited to create an effective and efficient but specific partitioning strategy.

We therefore also need to devise methods to make the choice of the algorithm and the parameters of the algorithms (e.g. block size, buffer groups) adaptive to the characteristics of the data, for instance to the data distribution, to the distribution of the arrival and to the distribution of the inter-arrival time, possibly with asymmetric characteristics of the data sets.

References

[1] R-tree portal, http://www.rtreeportal.org/datasets.html.

[2] A. N. Wilschut and P. M. G. Apers . Dataflow query execution in a parallel main-memory environment. In *Proc. First Intl. Conf. on Parallel and Distributed Info. Sys. (PDIS)*, pages 68–77, 1991.

[3] L. A. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter. Scalable sweeping-based spatial join. In *Proc. 24th Intl. Conf. Very Large Data Bases, VLDB*, pages 570–581, 24–27 1998.

[4] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 322–331, 1990.

[5] T. Brinkhoff, H.-P. Kkriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *ACM SIGMOD Intl. Conf. on Management of Data*, May 1993.

[6] T. Brinkhoff, H.-P. Kriegel, R. Schneider, and B. Seeger. Multi-step processing of spatial joins. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 197–208, 1994.

[7] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Parallel processing of spatial joins using r-trees. In *Proceedings of 19th Intl. Conf. on Data Engineering (ICDE)*, 1996.

[8] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD Intl. Conf. on Management of Data*, Aug 1984.

[9] P. J. Haas and J. M. Hellerstein. Ripple join for online aggregation. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, Philadelphia*, pages 287–298, 1999.

[10] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, and P. J. H. T. Roth. Interactive data analysis: The control project. In *IEEE Computer, 32(8):51-59*, August 1999.

[11] E. G. Hoel and H. Samet. A qualitative comparison study of data structures for large linear segment databases. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 205–214, 1992.

[12] Y. W. Huang, N. Jing, and E. Rundensteiner. Spatial joins using r-trees: Breadth-first traversal with global optimizations. In *Proc. 23th Intl. Conf. on Very Large Data Bases, VLDB*, pages 396–405, 1997.

[13] Z. G. Ives, D. Florescu, M. Fiedman, A. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, Philadelphia*, 1999.

[14] N. Kabra and D. J. Dewitt. Efficient mid-query reoptimization of sub-optimal query execution plans. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, Seattle*, pages 106–117, 1998.

[15] M.-L. Lo and C. V. Ravishankar. Spatial joins using seeded trees. In *ACM SIGMOD Intl. Conf. on Management of Data*, 1994.

[16] M.-L. Lo and C. V. Ravishankar. Spatial hash-joins. In *ACM SIGMOD Intl. Conf. on Management of Data*, May 1996.

[17] G. Luo, J. F. Naughton, and C. J. Ellmann. A non-blocking parallel spatial join algorithm. In *Intl. Conf. on Data Engineering*, pages 697–705, 2002.

[18] N. Mamoulis and D. Papadias. Integration of spatial join algorithms for joining multiple inputs. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 1–12, 1999.

[19] J. M. Patel and D. J. DeWitt. Partition based spatial-merge join. In *ACM SIGMOD Intl. Conf. on Management of Data*, May 1996.

[20] R. Ramakrishnan and J. Gehrke. *Database Management Systems, Third Edition*. McGraw-Hill, 2003.

[21] D. Rotem. Spatial join indices. In *Proceedings of 19th Intl. Conf. on Data Engineering (ICDE)*, pages 500–509. IEEE Computer Society, 1991.

[22] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley, MA, 1989.

[23] T. Sellis, N. Roussopoulos, and C. Faloutsos. R+-tree: A dynamic index for multi-dimensional objects. In *IEEE International Conf. on Very Large Databases*, 1987.

[24] T. Urhan and M. J. Franklin. XJoin: Getting fast answers from slow and bursty networks. Technical Report CS-TR-3994, Computer Science Department, University of Maryland, 1999.

[25] T. Urhan, M. J. Franklin, and L. Amsaleg. Cost-based query scrambling for initial delays. pages 130–141, 1998.

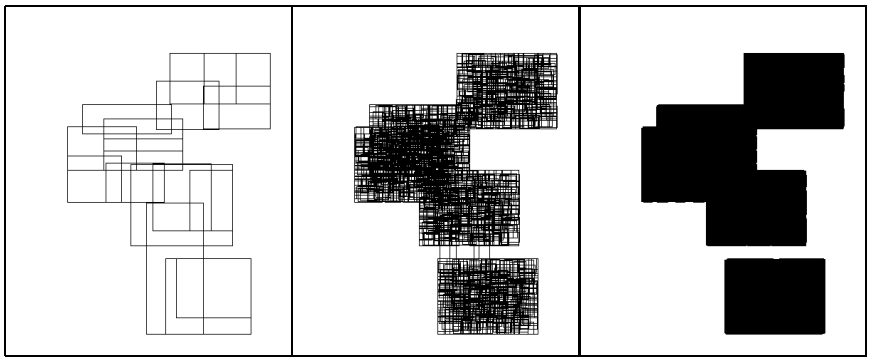


Figure 1. R-tree layout for R100C5

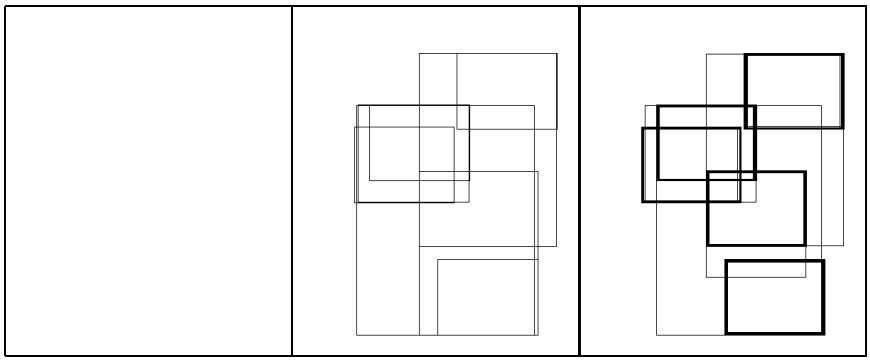
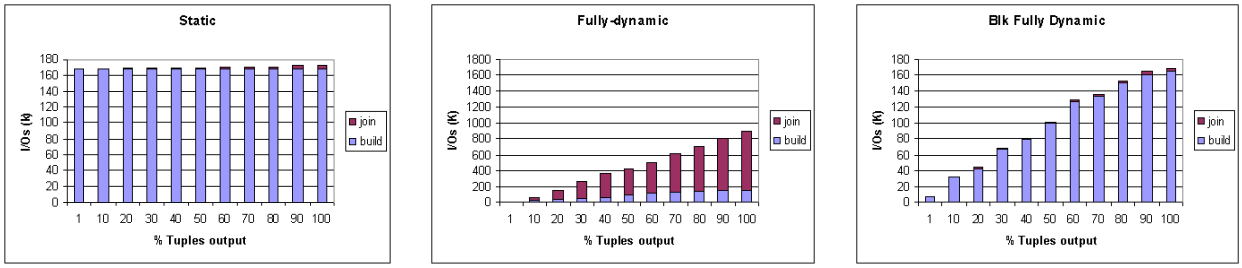
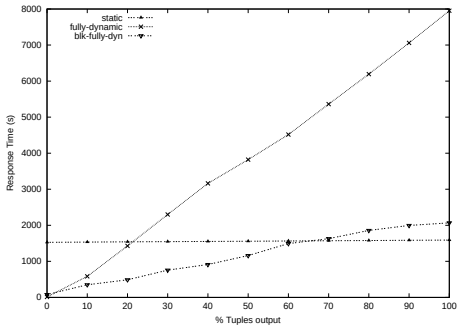


Figure 2. Summary R-tree layout for R100C5

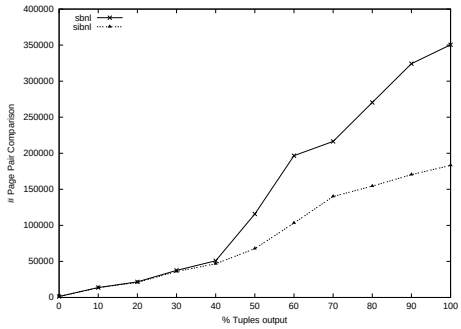


(a) Static (b) Fully Dynamic (c) Block Fully Dynamic

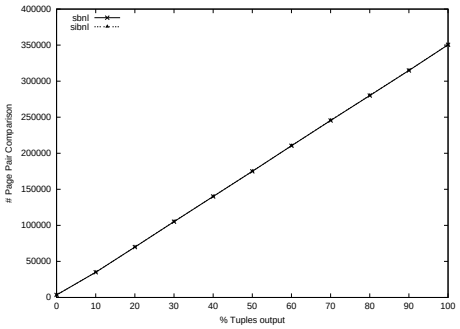


(d) Response Time

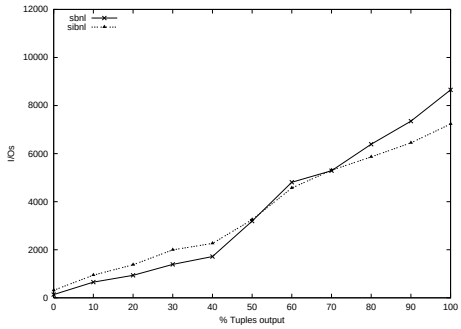
Figure 3. Comparison of R-tree Based Spatial Joins (R100KC5 ⋈ S100KC5)



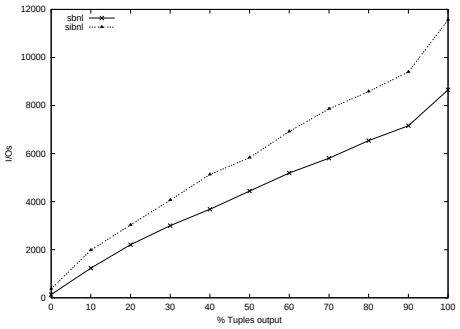
(a) Page Comparison
(Clustered)



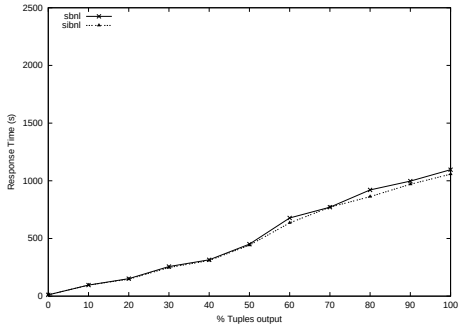
(d) Page Comparison
(Shuffled)



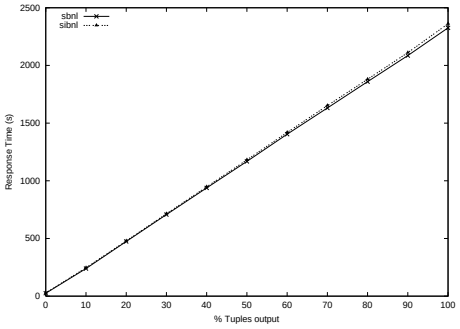
(b) I/Os
(Clustered)



(e) I/Os
(Shuffled)

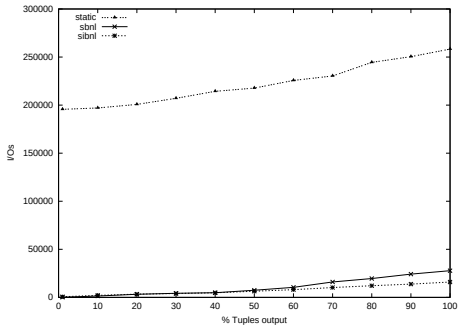


(c) Response Time
(Clustered)

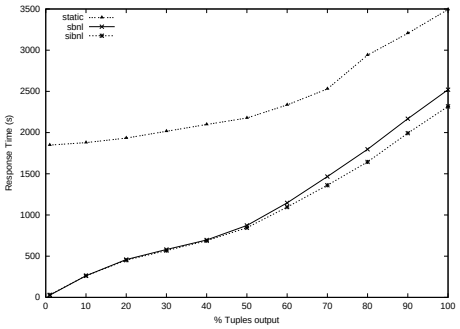


(f) Time
(Shuffled)

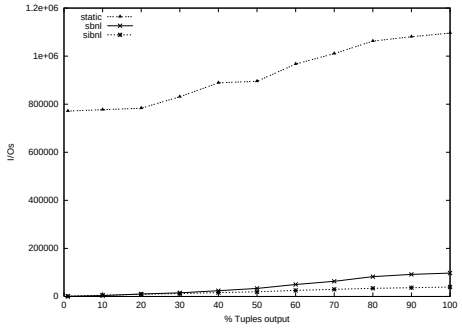
Figure 5. Clustered vs Shuffled (R100KC5 ⋈ S100KC5)



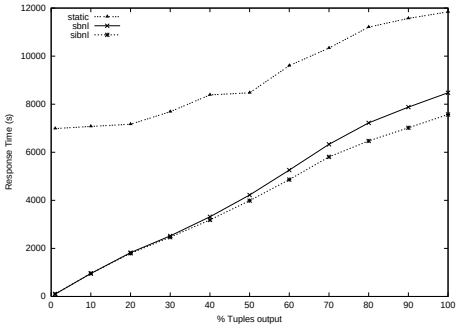
(a) I/O (R200KC5 ⋈ S200KC5)



(b) Response Time (R200KC5 ⋈ S200KC5)

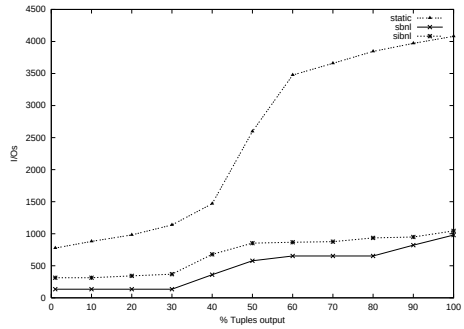


(c) I/O (R400KC5 ⋈ S400KC5)

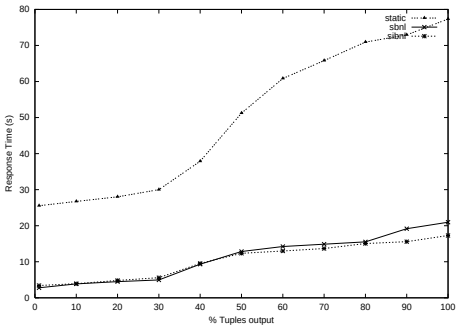


(d) Response Time (R400KC5 ⋈ S400KC5)

Figure 6. Scalability Test

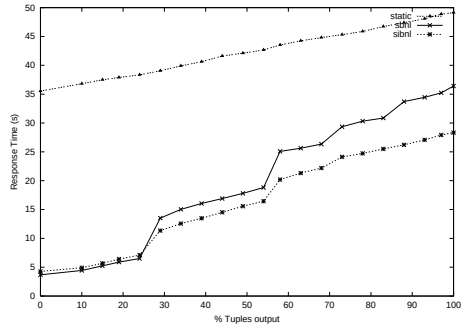


(a) I/Os (Greece)

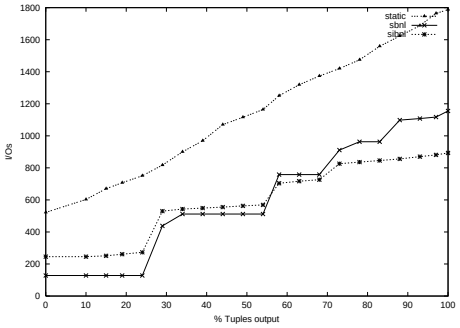


(b) Response Time (Greece)

(Rivers ⋈ Roads)



(c) I/Os (Germany)



(d) Response Time (Germany)

(Railroad Lines ⋈ Roads)

Figure 7. Performance on Real-Life Data Sets