In-Memory Spatial Join: The Data Matters!

Sadegh Nobari[†], Qiang Qu[†], Christian S. Jensen[‡]
{nobari,qiang}@siat.ac.cn, csj@cs.aau.dk
[†]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
[‡]Department of Computer Science, Aalborg University

ABSTRACT

A spatial join computes all pairs of spatial objects in two data sets satisfying a distance constraint. An increasing demand in applications ranging from human brain analysis to transportation data analysis motivates studies on designing new in-memory spatial join algorithms. Among recent proposals, the following six algorithms can efficiently perform in-memory spatial joins: *Size Separation Spatial Join* (S3), *Spatial Grid Hash join* (SGrid), *TOUCH*, *Partition Based Spatial-Merge Join* (PBSM), *Plane-Sweep Join* (PS), and *Nested-Loop Join* (NL).

This paper addresses the need for studies of aspects that may influence the performance of spatial join algorithms. In particular, given two datasets, A and B, the following aspects may affect performance: the datasets being real or synthetic data, the distributions with respect to density and location of the datasets, and the order of performing the spatial join ($A \bowtie B$ or $B \bowtie A$). To study the effects on performance of these aspects, we implement the six spatial join algorithms in a single framework and conduct extensive experiments.

The findings show that the data being real or synthetic, the data distribution, and the join order can influence substantially the performance of the algorithms. We present detailed findings that offer insight into different facets of each algorithm and that enable comparison across algorithms and datasets. Furthermore, we provide advice on choosing among the spatial join algorithms based on the empirical evaluation.

1. OBJECTIVES

Joins are important in many applications [3, 12, 13]. Spatial joins find spatially close object pairs in two data sets. Spatial joins are employed in many applications, and their use in-memory is becoming increasingly important [9, 13, 14] for two reasons. First, main memory has grown so large that many datasets fit main memory so that spatial joins can be performed entirely in-memory. Second, in-memory join is an unavoidable component of any join since, regardless of whether disk-based or in-memory joins are used, at least some of the join occurs in-memory. Several spatial join algorithms exist. Among them, the following 6 algorithms can efficiently perform in-memory spatial joins: 1) *Size Separation Spatial Join* (S3) [2], an algorithm based on a hierarchy of equi-width grids of increasing granularity; 2) *Spatial Grid Hash join* (SGrid) [4], a sampling algorithm that speeds up building an *R*-Tree on one dataset; 3) *TOUCH* [9], a hierarchical data-oriented space partitioning in-memory algorithm; 4) *Partition Based Spatial-Merge Join* (PBSM) [10], a multiple assignment algorithm that assigns each spatial object to all partitions it overlaps with; 5) *Plane-Sweep Join* (PS) [11], an algorithm that sorts the datasets in one dimension and scans the datasets synchronously; and 6) *Nested-Loop Join* (NL) [5] that iterates over both spatial datasets in a nested loop and compares all pairs of objects.

Despite many studies on spatial joins, an investigation is missing that studies the importance of the joining data itself. Given two datasets, our empirical study shows that when joining real and synthetic datasets, the data distribution and the order in which the datasets are joined (*join order*) can impact the performance of the algorithms considered very substantially. Our objectives are thus two-fold. First, we implement all the above algorithms in a unified and efficient in-memory spatial join framework in `C++` for ease of comparison and consistent reporting of results. In our framework, we apply implementation optimization and modularity wherever possible. Second, we aim to study how real versus synthetic data, data distribution, and join order influence spatial join performance, and how the findings suggest dirctions for designing robust, high-performance in-memory spatial join algorithms [1, 6, 7, 8, 15].

2. ALGORITHMS & IMPLEMENTATION

In the following, we briefly discuss the 6 spatial join algorithms we implemented in-memory. The *Nested Loop* (NL) join [5] compares all pairs of objects from both datasets exhaustively. The *Plane-Sweep* (PS) approach sorts the datasets based on an arbitrary dimension and scans both datasets synchronously. To perform PS, we employ a fast parallel sort to efficiently sort all the objects on one dimension. However, the performance of PS suffers from objects that are not near each other in the other dimensions. Despite its deficiencies, the plane-sweep approach is still broadly used for in-memory joins of the partitions resulting from disk-based spatial joins.

Disk-based approaches first partition both datasets and then join the resulting partitions in-memory via two different approaches, *multiple assignment* and *multiple matching.*
Multiple Assignment: this strategy assigns each spatial

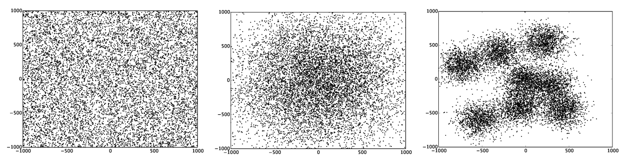

(a) Uniform (b) Gaussian (c) Clustered

Figure 1: Synthetic datasets

object to all partitions it overlaps with. PBSM [10] is an efficient multiple assignment approach that partitions the entire space of both datasets (A & B) into cells using a uniform grid. Then it joins the cells synchronously. SGrid on the other hand, constructs a grid only for the first dataset (A) and probes the intersecting cells in the constructed grid for each object of the other dataset (B).

Multiple Matching: this strategy that assigns each spatial object only to one of the partitions it overlaps with. S3 [2] is a multiple matching approach that maintains a hierarchy of L equi-width grids of increasing granularity. In D dimensions, the grid on a particular level l has $(2^l)^D$ grid cells and assigns each object of both datasets to a grid cell in the lowest level where it only overlaps one cell. Then S3 joins a cell with its counterpart as well as with cells on higher levels for all the cells.

The TOUCH algorithm consists of three steps: Tree construction, assignment, and batch join. TOUCH first builds an R-Tree from the objects of $dataset_A$, i.e., T_A. Then it assigns the objects of $dataset_B$ to the internal nodes of T_A. Finally, the algorithm performs the required pairwise comparisons according to the assignment of the objects of $dataset_B$ to T_A in the batch join step.

3. SETUP AND PRESENTATION

Configuration: The experiments are executed on a Linux Ubuntu 15.04 server equipped with 4 Intel Xeon E5-2650 v3 2.30GHz CPUs and 128GB RAM.

Data description: Our empirical study is done on two *categories* of datasets, namely real (i.e., do not follow any particular distribution) and synthetic. Each synthetic dataset is generated with varying *distributions* of 128,000 cuboids.

The distributions are *Uniform* (Figure 1(a)) (denoted as U), *Gaussian* (Figure 1(b)) (denoted as G), and *Clustered* (Figure 1(c)) (denoted as C) in a universe with a range of [-1000,1000] units per dimension. Figure 1 shows a 2D projection of the distributions. *Uniform* randomly assigns object centers in the bounded universe. *Gaussian* has a mean in the center of universe (0,0,0) and a dispersion of 1000 units. *Clustered* contains 10 clusters. Objects are randomly assigned to one of the cluster centers in the space, and each cluster has a Gaussian distribution ($\sigma = 200$) around the center of the cluster. Each dataset consists only of cuboid objects. After centers of cuboids are distributed, as explained above, each object size is randomly increased uniformly and independently, i.e., the correlation between dimensions is zero. We created 128 thousand objects with average mean length of 10 units.

In addition, to generate the synthetic datasets, we take 10 random samples of 128,000 cylinders (termed R) from Brain datasets (Axons and Dendrites) [9], which do not follow any particular distribution.

To examine the impact of the join order and distribution, we generate (or sample) 20 datasets for each distribution (category). Then we divide the datasets for each distribution into two groups A and B, each with 10 datasets of 128,000 objects. Finally, for each possible combination and order of the datasets, we run all six algorithms and measure the total time. The total time consists of loading time, processing time, and the time for maintaining a result. All the reported total times are averaged over 5 runs under the above-described configuration. In the figures, we distinguish join order (dataset A or B) and the dataset distribution (R, U, C, or G) in the legends. For instance $R_A \bowtie U_B$ means the left hand side dataset is a Real dataset and the right hand side is a Uniform dataset.

Result presentation: We run all 6 algorithms, namely S3, SGrid, TOUCH, PBSM, PS, and NL and report the results in the next section. Since NL was substantially slower than the other algorithms, with a total time of more than 5 hours, with omit NL from the charts. For sake of clarity, we present the results of running the remaining 5 algorithms in the following forms: Since SGrid consistantly performs slower than TOUCH and PBSM and faster than S3 and PS, we present the results for SGrid in Table 1 and the results for the other four algorithms in Figures 1 to 7; In these figures, since TOUCH and PBSM are always faster than S3 and PS, we create two groups (TOUCH and PBSM in one group, S3 and PS on the other group) and use different vertical axes for each group.

4. RESULTS

Our results are summarized as follows:

- The performance of NL and PS is insensitive to the join order.
- When one of the datasets follows a particular distribution or when joining two datasets of the same category, the performance of SGrid is not affected by swapping the datasets. Furthermore, when joining two datasets of different categories, the performance of SGrid varies very substantially when swapping the join order.
- The performance of S3, TOUCH, and PBSM changes when swapping the join order and changing the dataset category of the arguments. As a result, there is not single winner among these algorithms. However, TOUCH does outperform the algorithms in most cases, although in few scenarios, TOUCH can be outperformed by PBSM for some join orders and data distributions. For instance, when joining R and U, the join order of join is critical to the performance of TOUCH. In one order, i.e., $U_B \bowtie R_A$, TOUCH outperforms PBSM while we swap the order, i.e., to $R_A \bowtie U_B$, PBSM outperforms TOUCH.

We proceed to provide detailed observations of each impact separately and finally cover all the impacts together.

4.1 Join order (*order*)

We first observe the impact of swapping the order of the datasets participating in a join, i.e., the *join order*. To observe this impact, we join different samples of the same category of dataset, i.e. real or synthetic, while swapping the join order. The join order has the least impact on PS, while S3 is affected the most. While affected by the join order, TOUCH outperforms all the other algorithms when joining datasets of the same category and distribution.

In Table 1, rows with same dataset, i.e., A=B, tell that SGrid is unaffected by the join order when keeping the dataset category and distribution unchanged. The results of this set of experiments while we join datasets of the same category

Dataset		Total time (s)	
A	B	A ⋈ B	B ⋈ A
Real	Real	142.517	141.37
	Uniform	16.799	48.449
	Gaussian	1.906	2.740
	Clustered	9.662	25.526
Uniform	Uniform	171.782	171.911
	Gaussian	3.579	3.746
	Clustered	41.541	41.196
Gaussian	Gaussian	3.245	3.243
	Clustered	3.450	3.606
Clustered	Clustered	53.706	53.684

Table 1: SGrid performance for varying datasets.

and distribution show that the join order does not affect the relative performance of SGrid. The study thus offers evidence that the join order does not matter when the category and distribution of the datasets are fixed.

4.2 Joining real with synthetic datasets (*category*)

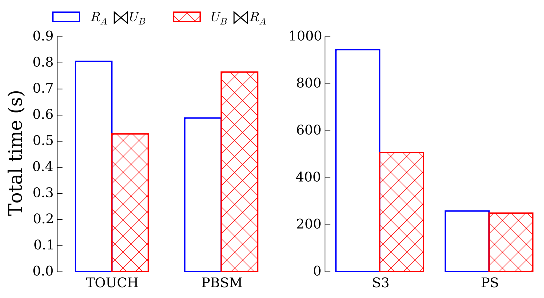
Having observed the impact of changing the join order of datasets of the same category and distribution, we proceed to observe how join order is influenced when we join a real dataset with varying synthetic datasets. Thus, we want to observe how the performance of the join algorithms is affected when one dataset is not following a particular distribution while the other dataset has a particular distribution. Figures 2, 3, and 4 and Table 1 contain results of joining Real (R) dataset with Uniform (U), Clustered (C) and Gaussian (G) datasets, respectively. The performance of all algorithms show a dependence the join order when swapping the category of datasets. The algorithms are affected to the extent that their relative performance changes so that there is no clear winner.

TOUCH and PBSM, in Figures 2 and 4, exhibit an interesting change in their relative performance, so that TOUCH can outperform PBSM only when TOUCH constructs its hierarchical data structure based on the uniform dataset for $U \bowtie R$ and based on the Real dataset for $R \bowtie G$. In all other cases, PBSM outperforms TOUCH. This behavior of TOUCH shows how this algorithm exploits the distribution of the objects. Indeed, TOUCH can outpeform the other algorithms when its constructed tree based on the two datasets is balanced, meaning that similar numbers of objects from both datasets occur in its tree nodes. For instance, when joining Guassian and real datasets, the real dataset can accommodate a balanced distribution of objects of both datasets in the tree constructed by TOUCH. And when joining real and clustered datasets, none of the datasets can yield a balanced tree because the datasets are not mutually aligned in terms of the density of their objects' distribution.

SGrid (Table 1 rows 2, 3, and 4) shows a substantial change, about 3×, in performance when swapping the join order. This indicates that all the algorithms, except the quadratic comparison (NL), are influenced by join order when dataset categories are different. Therefore, it is essential to carefully select the join order when datasets are not following a particular pattern of object distribution, e.g., when joining real datasets.

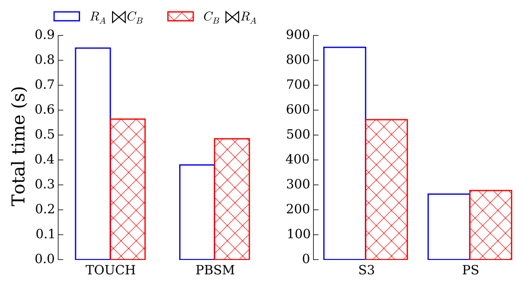
4.3 Joining synthetic datasets (*distribution*)

Finally, we put focus on joining synthetic datasets of different distributions. In other words, we join datasets that



(a) R ⋈ U (b) U ⋈ R

Figure 2: Real (R) ⋈ Uniform (U) and vice versa.



(a) R ⋈ C (b) C ⋈ R

Figure 3: Real (R) ⋈ Clustered (C) and vice versa.

follow a particular distribution, namely U, G, and C, while the pattern is different for each. Figures 5, 6, and 7 are results of joining Uniform (U) and Clustered (C) datasets, Uniform (U) and Gaussian (G) datasets, and Clustered (C) and Gaussian (G) datasets, respectively. And rows 6, 7, and 9 in Table 1 show the performance of SGrid for this set of experiments. All algorithms show less impact on the join order when each dataset follows a distribution, in contrast to real datasets with no particular distribution pattern. TOUCH outperforms all the algorithms and it shows a larger change of its performance for different join orders when none of the datasets have a uniform distribution.

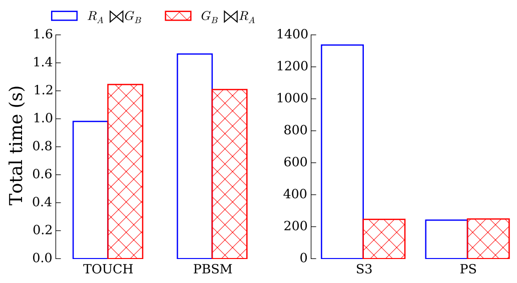
NL is always steadily slower than all the other 5 algorithms, no matter what join order or data distribution we use. For any two of our datasets, NL always requires close to 5 hours to perform the join.

4.4 Overall comparison

The results in Table 1 suggest that when each of the joining datasets follows a particular distribution, the performance of SGrid is not affected when swapping the datasets, i.e., constructing the grid based on the first dataset or the second makes little difference. However, when joining a real dataset with a synthetic dataset, the join order affects the performance of SGrid. This behavior can be explained by the number of cells that an object intersects with as well as the number of objects in the intersecting cells, e.g., the density of the cell.

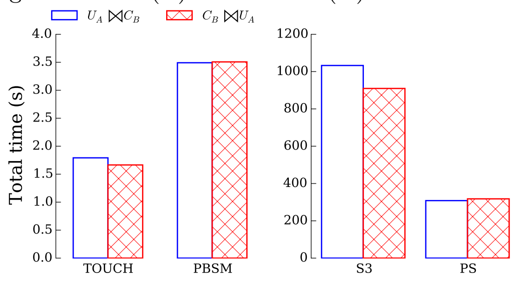
All presented results suggest that the most challenging distribution for all algorithms is that of the Real (R) dataset. The reason is possibly the irregular density/distribution of the objects all over the universe. However, TOUCH and PBSM generally outperform all the other joins due to their ability to handling varying densities.

Among all the algorithms considered, S3 is the most sensitive to the join order, especially when none of the datasets are neither uniform nor real. When both datasets are either uniform or real, the density of objects is consistent with the



(a) R ⋈ G (b) G ⋈ R

Figure 4: Real (R) ⋈ Gaussian (G) and vice versa.



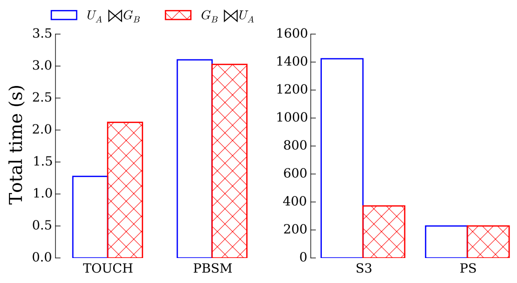
(a) U ⋈ C (b) C ⋈ U

Figure 5: Uniform (U) ⋈ Clustered (C) and vice versa.

region, meaning that similar regions of both datasets have similar object densities. The sensitivity of S3 arises from its dependency on its hierarchical model. Its hierarchy is constructed for the two datasets independently. Therefore, when the order of join changes, the algorithm is affected substantially. In contrast to S3, NL and PS are algorithms that are the least sensitive to the join order. The reasons are that NL uses quadratic comparison to enumerate the search space and that PS sweeps the space one dimension at a time.

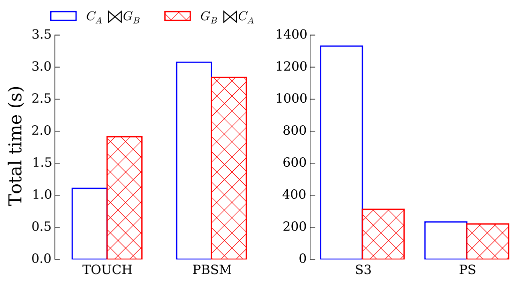
From a practical point of view, TOUCH, SGrid, and PBSM are generally faster than the other algorithms when joining large datasets. SGrid is preferred when the datasets are both real or synthetic with the same distribution. Next, when the datasets are not both real or synthetic with the same distribution, like real and clustered, PBSM is preferred because TOUCH is sensitive to the join order. TOUCH is the fastest when the datasets follow the same distribution. The findings of this paper can help when designing practical algorithms for multi-joins (e.g., $(A \bowtie B) \bowtie C$) and multiple spatial joins (e.g., joining A, B, and C simultaneously). In such algorithms, the order and distribution play even more crucial roles than what we observed in these experiments.

This study suggests that without prior knowledge of the datasets, there is no single winner among the spatial join algorithms. Further, obtaining such knowledge is not always feasible or possible, due to the associated cost or the real-time nature of the datasets, e.g. streaming, intermediate, or growing datasets. Therefore, designing a spatial join algorithm that gradually adapts to the distribution of the argument datasets during the processing and that is capable of changing the order of the joining datasets internally are important for designing robust and scalable spatial join algorithms.



(a) U ⋈ G (b) G ⋈ U

Figure 6: Uniform(U) ⋈ Gaussian(G) and vice versa.



(a) G ⋈ C (b) C ⋈ G

Figure 7: Gaussian(G) ⋈ Clustered(C) and vice versa.

5. REFERENCES

[1] A. Hadian, S. Nobari, B. Minaei-Bidgoli, and Q. Qu. ROLL: fast in-memory generation of gigantic scale-free networks. In *SIGMOD*, pages 1829–1842, 2016.

[2] N. Koudas and K. C. Sevcik. Size Separation Spatial Join. In *SIGMOD '97*.

[3] S. Liu, Q. Qu, and S. Wang. Rationality analytics from trajectories. *TKDD*, 10(1):10:1–10:22, 2015.

[4] M.-L. Lo and C. V. Ravishankar. Spatial Hash-Joins. In *SIGMOD*, pages 247–258, 1996.

[5] P. Mishra and M. H. Eich. Join Processing in Relational Databases. *ACM Computing Surveys*, 24(1):63–113, 1992.

[6] S. Nobari, T. Cao, P. Karras, and S. Bressan. Scalable parallel minimum spanning forest computation. In *PPOPP*, pages 205–214, 2012.

[7] S. Nobari, P. Karras, H. Pang, and S. Bressan. L-opacity: Linkage-aware graph anonymization. In *EDBT*, pages 583–594, 2014.

[8] S. Nobari, X. Lu, P. Karras, and S. Bressan. Fast random graph generation. In *EDBT*, pages 331–342, 2011.

[9] S. Nobari, F. Tauheed, T. Heinis, P. Karras, S. Bressan, and A. Ailamaki. TOUCH: in-memory spatial join by hierarchical data-oriented partitioning. In *SIGMOD*, pages 701–712, 2013.

[10] J. M. Patel and D. J. DeWitt. Partition Based Spatial-Merge Join. In *SIGMOD*, pages 259–270, 1996.

[11] F. Preparata and M. Shamos. *Computational Geometry: An Introduction.* Springer, 1993.

[12] P. Roy, J. Teubner, and R. Gemulla. Low-latency handshake join. *PVLDB*, 7(9):709–720, 2014.

[13] D. Sidlauskas and C. S. Jensen. Spatial joins in main memory: Implementation matters! *PVLDB*, 8(1):97–100, 2014.

[14] B. Sowell, M. A. V. Salles, T. Cao, A. J. Demers, and J. Gehrke. An experimental analysis of iterated spatial joins in main memory. *PVLDB*, 6(14):1882–1893, 2013.

[15] F. Tauheed, S. Nobari, L. Biveinis, T. Heinis, and A. Ailamaki. Computational neuroscience breakthroughs through innovative data management. In *ADBIS*, pages 14–27, 2013.