Spatial Joins: What's next?

Panagiotis Bouros
Institute of Computer Science
Johannes Gutenberg University Mainz, Germany
bouros@uni-mainz.de

Nikos Mamoulis
Dept. of Computer Science and Engineering
University of Ioannina, Greece
nikos@cs.uoi.gr

Abstract

The spatial join is a popular operation in spatial database systems and its evaluation is a well-studied problem. This paper reviews research and recent trends on spatial join evaluation. The complexity of different data types, the consideration of different join predicates, the use of modern commodity hardware, and support for parallel processing open the road to a number of interesting directions for future research, some of which we outline in the paper.

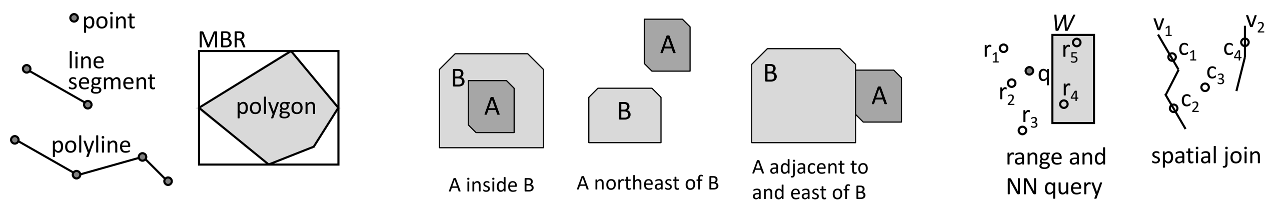
1 Introduction

Spatial data are nowadays more ubiquitous than ever before; examples of such data include meteorological maps, biological and scientific data, socio-economic data, agricultural data and geo-tagged social media. Thanks to the proliferation of mobile location-aware devices and services (e.g., smartphones, tablets, wearables, GPS devices, mobile applications, satellites), the volume of spatial data generated every single day increases at a staggering and unprecedented rate, and so does the number of applications and domains where such data are collected and analyzed. The challenges and the importance of big spatial data management have been extensively sung, e.g., in [14, 19, 48].

The *spatial join* is a fundamental data operation. Traditionally, such a join finds application in spatial data management systems [16] and Geographic Information Systems (GIS) [24]. GIS, for example, typically store multiple thematic layers (e.g., road network, hydrography), which are spatially joined in order to find object pairs (e.g., roads and rivers) that intersect. In addition, spatial joins are used to support data mining operations such as clustering [15] and pattern detection [11]. Today, spatial joins play a key role also in scientific applications. For instance, they can determine neuron synapses in brain models, i.e., pairs of neuron branches that are within very small distance to each other [27], while in medical imaging, spatial joins are used to determine proximity of cells and to facilitate the analysis of high resolution images of human tissue specimens for more effective diagnosis, prediction and treatment of diseases [3].

In this article, we focus on the most common definition of spatial joins termed the *spatial intersection* join. The goal is to determine pairs of spatial objects, e.g., (road, river) pairs, that have at least one common point in space. Besides its popularity and wide adoption, note that evaluation methods for the spatial intersection join can be also used for other types of spatial joins such the *spatial distance* and the *nearest neighbor* joins.

A wide range of spatial join algorithms have been proposed in the literature [18, 25]. Most of them assume that the input data are disk-based and their objective is to minimize I/O accesses during the join. Given the fact that main memories are constantly becoming bigger, cheaper and faster, in-memory join processing has recently received significant attention [29]. In addition, given that commodity hardware supports parallel processing, multi-core join evaluation has also been the focus of recent research. In line with this trend, this article focuses on parallel in-memory evaluation of spatial joins on modern hardware. Our goal is to describe the landscape of efficient spatial join computation under this prism and to discuss interesting directions for future research.



(a) data types (b) spatial relationships (c) spatial queries

Figure 1: Examples of spatial data, relationships and queries

2 Background

In this section, we provide a brief background on spatial data management [26], discussing basic data and query types, indexing structures and principles for query evaluation.

Common types of spatial objects include the *point* (defined by one value per dimension), the *rectangle* (defined by one interval per dimension), the *line segment* (defined by a pair of points), the *polygon* (defined by a sequence of points), the *polyline* (defined by a sequence of points), etc. Figure 1(a) shows examples of these datatypes on the plane. To characterize the relative position between two spatial objects, three classes of *spatial relationships* can be used. *Topological* relationships (e.g., overlap, inside, contains, disjoint, etc.) model relationships between the geometric extents of objects. *Directional* relationships (e.g., north/south, east/west, above/below, left/right, etc.) compare the relative locations of the objects with respect to a coordinate (or cardinal) system. Last, *distance* relationships capture distance information between two objects. Figure 1(b) illustrates examples of spatial relationships.

The most frequently applied query operation on spatial data is the *spatial selection* (or *range query*) which asks for the objects that intersect (or are inside) a spatial range, or are within some distance from a reference spatial object. Another popular operation is the *nearest neighbor search* which asks for the objects that are the nearest to a reference location. Finally, the *spatial join* finds pairs of objects from two collections that qualify a spatial predicate. The most common join operation is the *spatial intersection join* which we primarily discuss in this article. Formally, given collections R and S, the objective is to find pairs (r, s), such that $r \in R$, $s \in S$ and their geometries intersect, i.e., r and s have at least one common point. Other popular definitions of spatial joins include the *ϵ-distance join* which determines (r, s) pairs of objects within at most distance ϵ from each other, and the *nearest-neighbor* join which returns for every object $r \in R$, the nearest objects from input collection S. As an example of a spatial selection assume that r_1 to r_5 in Figure 1(c) are locations of restaurants and that a user is interested in finding which restaurants are located in region W. The result includes objects r_4 and r_5. If the objective is to find the nearest restaurant to location q, the result is r_2. Assuming that v_1 and v_2 are rivers and c_1 to c_4 are cities, the result of the spatial intersection joins between these two sets are pairs $\{(v_1, c_1), (v_1, c_2), (v_2, c_4)\}$.

Due to the potentially complex geometry of the objects, spatial query evaluation is typically performed in two steps. In the *filter* step, the query is applied on the *minimun bounding rectangles* (MBRs) of the objects, which are simple lightweight approximations; see for example the MBR of a polygon in Figure 1(a). Object MBRs (or pairs of object MBRs) that do not qualify the query can be pruned in the filter step, while for the objects (or object pairs) that pass the filter an expensive *refinement step* is applied using their exact geometries. For example, if the MBR of a river does not intersect the MBR of a city, there is no chance that the exact geometries of the two objects intersect.

In order to process spatial queries efficiently, a number of spatial access methods have been proposed to index the spatial object collections. The most popular spatial index is the R-tree and its variants [17, 6], a balanced tree, similar to the B$^+$-tree, which groups nearby object MBRs into the leaf nodes of the tree. The non-leaf nodes are formed by hierarchically grouping nearby MBRs of lower-level nodes. When evaluating a spatial query, a node (and the corresponding sub-tree) whose MBR does not qualify the predicate of the query can be pruned, drastically reducing the search space. A simpler index for memory-resident data is a spatial grid, which divides the space into cells and its more sophisticated quadtree version [41].

3 Join Computation Landscape

Given two large collections of spatial objects R and S, a spatial join is in principle processed by first dividing the collections into small partitions and then joining these partitions. For each collection, we may use an existing partitioning or index; in either case, the join is broken down into a number of small joins that can be processed fast in memory. In what follows, we overview the landscape in computing spatial join queries and organize literature in four key categories. Note that the vast majority of the proposed evaluation methods focus on the filter step.

3.1 Small Join Computation

For in-memory processing of small spatial joins, a typical approach is to use adaptations of a plane sweep algorithm that compute rectangle intersections [37]. The most commonly used adaptation was proposed by Brinkhoff et al. in [10]. In brief, the collections are first sorted based on their lowest value in one dimension; then, the sorted inputs are scanned concurrently and merged in a merge-join fashion. This process resembles a perpendicular line that sweeps along the sorting dimension. Every time the sweep line stops, e.g., at object r from R, the algorithm *forwardly* scans input S to produce the join results. Arge et al. studied in [5] a variation of the plane sweep algorithm which maintains an active list of previously encountered objects at every position of the sweep line, performing essentially a *backward scan*. In practice, the performance gain over [10] is insignificant unless a special data structure is used to organize active lists, similar to the gapless hash map of [35].

3.2 Data Partitioning

Data partitioning has been considered as a *divide-and-conquer* approach to split the input collections into smaller subsets that can then be spatially joined fast in memory. A partition from input R is then joined with a partition from S only if their MBRs intersect. A large number of spatial join algorithms that follow this paradigm have been proposed; the methods can be classified into *single-assignment*, multi-join (SAMJ) methods and *multi-assignment*, single-join (MASJ) approaches [23]. SAMJ methods assign each object to exactly one partition; the partitions are determined by spatial clustering heuristics. A partition from one input may have to be joined with multiple partitions of the other. The *R-tree Join* algorithm (RJ) in [10] is a classic SAMJ approach when both inputs are indexed by an R-tree. RJ starts by finding all pairs of entries (e_R, e_S) one from each root node of the trees that intersect. For each such pair, the algorithm recursively applies the same procedure for the nodes pointed by e_R and e_S, until pairs of leaf node entries (which correspond to intersecting object MBRs) are found. A SAMJ approach that does not rely on pre-defined indices is *Size Separation Spatial Join* from [20].

On the other hand, the borders of the partitions in MASJ are pre-determined, and so an object is assigned to every partition it spatially intersects. Each partition from one collection is joined with exactly one partition from the other (which has exactly the same MBR). The most popular MASJ approach is *Partition-based Spatial Merge Join* (PBSM) [32] which divides the space by a regular grid and assigns objects from both input collections to all tiles that spatially overlap them. For each partition, PBSM accesses the objects from both inputs and performs the small join in memory (e.g., using plane sweep). Since an object can be replicated to multiple tiles, this method can generate duplicate results; duplicates can be eliminated by reporting a join result in a tile only if a specific corner of the intersection falls in the tile [12]. Other MASJ approaches include *Spatial Hash Join* from [23] and *Scalable Sweeping-Based Spatial Join* from [5].

More recent work investigates the role of the distribution and density of the input collections. Motivated by a neuroscience application, which requires joining datasets of contrasting density, Pavlovic et al. design a spatial join algorithm in [34] that partitions the dense dataset. Then, the algorithm ‘crawls’ through the partitions guided by the object locations in the sparse dataset, skipping partitions that do produce any results. Based on the same motivation, a more sophisticated approach in [33] called *TRANSFORMERS*, adapts the type of partitioning (MASJ or SAMJ) and the join technique used locally, depending on the differences in the densities of the two inputs.

3.3 In-Memory Processing

Despite the recent advances that allow us to store the entire input collections in main memory, directly applying plane sweep can be too expensive. This is due to the large number of candidates produced by forward scans, which do not materialize to actual results. Hence, in-memory join approaches also consider data partitioning or indexing to accelerate the join computation. For example, a grid (similar to PBSM) can be used to break the problem into numerous small instances that can be solved fast. The *TOUCH* algorithm from [30] is an in-memory algorithm, designed for scientific applications that join huge datasets that have different density and skew. TOUCH first bulk-loads an R-tree for one of the inputs using the STR technique [21]. Then, all objects from the second input are assigned to buckets corresponding to the non-leaf nodes of the tree. Each object is hashed to the lowest tree node, whose MBR overlaps it, but no other nodes at the same tree level do. Finally, each bucket is joined with the subtree rooted at the corresponding node with the help of a dynamically created grid data structure for the subtree. A recent comparison of spatial join algorithms for in-memory data [29] shows that PBSM and TOUCH perform best and that the join cost depends on the data density and distribution. Tauheed et al. [45] suggest an analytical model for configuring the grid of PBSM-like join processing in main memory.

3.4 Distributed and Parallel Processing

Early efforts on parallelizing spatial joins include extensions of the R-tree join and PBSM algorithms in a distributed environment of single-core processing nodes with local storage. For R-tree join, overlapping pairs of root entries essentially define independent join processes on the corresponding sub-trees. Hence, these tasks can be assigned to different computer nodes [9]. To reduce I/Os, a virtual global buffer is shared among the processing nodes to avoid accessing the same data multiple times from the disk. An early approach in parallelizing PBSM was presented in [54]; the method asks the processing nodes to perform the partitioning of data into tiles independently and in parallel. Then, each computer node is assigned one partition and then nodes exchange data, so that each one gets all objects that fall in its partition. The join phase is finally performed in parallel.

Recently, the research interest shifted to spatial join processing for distributed cloud systems and multi-core processors. Pandey et al. conducted an experimental evaluation in [31] between parallel and distributed spatial data management systems where, among other queries, spatial joins were tested. Essentially, approaches in this context fall into two categories. The first category capitalizes on the popularity and commercial success of the MapReduce framework. For instance, the *Spatial Join with MapReduce* (SJMP) algorithm from [53] is an adaptation of the PBSM algorithm where grid tiles are examined in a space-filling curve order and grouped to partitions in a round-robin fashion. *Map* tasks assign input objects to one or more partitions based on the tiles they overlap, while every partition is then processed by a separate *reduce* task. The reducers perform their joins by dividing the space that corresponds to them into stripes and then applying plane sweep. Duplicate results are avoided by reporting a join pair only at the tile with the smallest id where the two objects commonly appear. *Handoop-GIS* [4] processes spatial joins in a similar manner, i.e., by applying a regular grid. A key difference is that the data objects are both globally and locally indexed. A *global index* shared between nodes, is used to find the HDFS files where the contents of each tile are stored. A *local index* is defined on every processing node to independently perform a spatial join. Finally, the results are merged. *Spatial-Hadoop* [13] also employs a global index, stored on the Master node. The system investigates the best join strategy when partitioning schemes pre-exist the queries for the input collections. If the two join inputs are partitioned differently essentially, we could either directly use the existing partitions and join every pair of overlapping partitions or re-partition the smaller input according to the partitioning of the larger and then use PBSM. The cost of each the two strategies is estimated and the cheapest one is selected accordingly. A query optimizer for MapReduce-based spatial join algorithms is presented in [40]. The second category of systems build on top of Apache Spark [51]. *Simba* [46], *SpatialSpark* [47], *GeoSpark* [49], *LocationSpark* [44] and *Magellan* [1] store both the indexing structures and the intermediate results in memory shared by all nodes in the cluster. Computing spatial joins is focused on effective spatial indexing of the *resilient distributed datasets* (RDDs).

4 Directions for Future Research

While there has been a vast amount of work on spatial join evaluation, the continuous evolution of hardware and the increase of volume and variety of big spatial data opens new, promising research directions. In this section, we briefly discuss research challenges and open issues related to spatial join evaluation.

4.1 Specialized join algorithms for different datatypes and accelerating the refinement step

Most spatial join algorithms treat all types of spatial data in the same way. Specifically, they solve the join problem on the MBRs of the objects in a *filter step* and then, for each pair of intersecting rectangles, they apply a *refinement step*. However, for certain (simple) data types, we can design specialized join algorithms that apply directly on the exact geometric representations of the objects, while achieving a cost similar to that of the MBR intersection join. In a recent work in this direction [28], a multi-core spatial join algorithm based on plane sweep was designed for line segment collections. A promising direction is to explore the development of algorithms that operate in the same spirit and handle other spatial datatypes such as polylines.

The refinement step of a spatial join can be much more expensive than the filter step if the objects have complex spatial extent. A number of ideas have been proposed in the direction of reducing the cost of the refinement step. In this direction, besides the MBR, alternative object approximations [8, 55] and *true hit filtering* can be applied. For example, objects can be approximated by the maximum rectangles or circles that are enclosed in them. If two such approximations intersect, then we can guarantee that the exact geometries of the objects intersect [8]. Another idea is to define a raster representation for each object, where a set of coarse pixels (i.e., cells of a fine grid) is used. For the cells that each object intersects, we can measure the percentage the cell that the object covers. If two objects intersect a cell by at least 50%, this guarantees that they are a spatial join result. An interesting research direction is to parallelize this technique (i.e., handle each cell in parallel) and the challenge would be to (i) handle duplicates, (ii) minimize the required space due to object replication.

Two recent pieces of related work study the point-to-polygon joins. Zacharatou et al. [50] use raster representations of the objects and employ GPUs to parallelize the spatial join, by handling each pixel in parallel. There is no issue of duplicate elimination because a point is guaranteed to intersect a polygon in at most one cell. Kipf et al. [19] evaluate point to polygon joins again using object decompositions (by a quadtree representation) and paralellization, aiming at minimizing the refinement cost. There is still room for improving the mechanisms for avoiding refinements wherever possible and for optimizing how refinements are implemented and scheduled in a distributed environment [39]. An interesting direction is to study polygon-to-polygon joins in parallel using raster or quadtree representations for each object. The main challenge is to avoid the production of duplicate results and the computational overhead for their computation.

4.2 Distance and nearest neighbor joins

Distance and nearest neighbor joins [52] are evaluated by extending or adapting algorithms for intersection joins. Special algorithms that take into consideration the fact that such joins are typically conducted between point-sets could be designed. For example, Bouros et al. study the computation of ϵ-distance joins in [7], where the objective is to find pairs of points within distance at most ϵ to each other. A regular grid, where the projections of the cells at each axis have length at most ϵ is used to online partition the data. Then, each cell needs to be joined with at most five cells in order to produce the result. Interesting research directions would be to parallelize this approach and to also consider objects which are not points. A first attempt towards the former was presented in [42] for distributed spatial data on top of MapReduce. For the point-to-polygon distance join, a combination of approaches from [50, 19] and [7] could be applied. Nearest neighbor joins are more challenging because the nearest neighbor of a point might not be located in the same or neighboring cells. Using statistics about the cell occupancies or a non-uniform (quadtree-style) partitioning could be a way for approaching this problem.

4.3 Scaling up vs. scaling out

In most of the applications that manage spatial data, the inputs can be preprocessed and then easily fit in the main memory of a single commodity machine. For example, in most public spatial data collections (see, for example, http://spatialhadoop.cs.umn.edu/datasets.html) the number of objects is in the order of 100M or less. Such data collections can fit in the memory of a commodity machine. Hence, scaling up (i.e., solving a medium size problem more efficiently) might be a more relevant problem compared to scaling out to larger data sizes that need large clusters or cloud computing algorithms. Multi-core parallelism is gaining ground, so a promising research direction is to design good shared-memory parallel algorithms for spatial joins, especially in applications where the data are produced and need to be processed at a high rate, i.e., streaming spatio-temporal data [19, 43]. Spatial join computation can also benefit from the high degree of parallelization achieved by using a single or multiple GPU chips [36], besides accelerating the refinement step. Aghajarian et al. present an attempt towards this direction in [2].

4.4 Optimizing partition-to-partition joins

As discussed already, spatial join algorithms for large-scale data typically operate in two phases. In the first phase, the data are partitioned and in the second phase the partitioned data are joined. If an spatial index exists for a join input, it can be used to define the partitions of the input, hence the partitioning phase for that input can be skipped. Although a large number of join algorithms have been proposed, the problems of (i) selecting the most appropriate partitioning and (ii) selecting the most appropriate technique for each partition-to-partition join has not been studied adequately. For (i), we need accurate cost models, based on spatial statistics, which can be used to determine the best way to partition the data. For (ii), a typical approach is to use plane sweep, however, this might not be efficient for joining partitions of contrasting skew and densities and for 3D data. In such cases, a finer partitioning can be used for the smaller partition to spatially hash its data and the objects in the larger partition can be probed against it [45, 33]. So far, there is no comprehensive evaluation on the conditions under which this spatial hashing partition-to-partition join technique is faster than plane sweep.

4.5 Extended join Operations

Spatial joins are traditionally defined and studied with respect to geometries or locations while ignoring other types of information attached to the input objects; an exception arises in the case of spatio-temporal joins where the temporal aspect is additionally taken into account. However, modern spatial data are not only more ubiquitous than ever but also more complex; they can be routinely assigned or associated with other types of information, e.g., text or social information. In this context, it is interesting to examine whether new join operations can be defined building on top of spatial joins and incorporating other types of information available. Besides defining these new join operations, we also need to investigate their efficient computation; a key challenged is how to combine the different algorithms and indexing structures employed for each type of information. Efforts towards this direction include the *spatio-textual similarity* join proposed in [7] which comes as hybrid of a spatial ϵ-distance join and a set similarity join, the techniques proposed in [22] for joins in GeoRDF data and the top-k distance join in [38] which uses the numerical information present in the objects to formulate the ranking component of the join.

5 Conclusions

In this article, we briefly reviewed evaluation techniques for spatial joins, with a focus on in-memory join processing and on parallel and distributed evaluation. Although spatial joins have been studied for decades, recently the research interest has been renewed, due to the opportunities brought by modern commodity hardware, which comes together with large memories and CPUs or GPUs with high parallelization capacity. This brings in chances and challenges for re-designing classic approaches for partitioning and join evaluation and accelerating the refinement step of the join, which we outline in the paper.

References

[1] Magellan: Geospatial analytics using spark. https://github.com/harsha2010/magellan.

[2] D. Aghajarian, S. Puri, and S. K. Prasad. GCMF: an efficient end-to-end spatial join system over large polygonal datasets on GPGPU platform. In *SIGSPATIAL*, pages 18:1–18:10, 2016.

[3] A. Aji, F. Wang, and J. H. Saltz. Towards building a high performance spatial query system for large scale medical imaging data. In *SIGSPATIAL/GIS*, pages 309–318, 2012.

[4] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. H. Saltz. Hadoop-GIS: A high performance spatial data warehousing system over mapreduce. *PVLDB*, 6(11):1009–1020, 2013.

[5] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter. Scalable sweeping-based spatial join. In *VLDB*, pages 570–581, 1998.

[6] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[7] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *PVLDB*, 6(1):1–12, 2012.

[8] T. Brinkhoff, H. Kriegel, R. Schneider, and B. Seeger. Multi-step processing of spatial joins. In *SIGMOD*, pages 197–208, 1994.

[9] T. Brinkhoff, H. Kriegel, and B. Seeger. Parallel processing of spatial joins using R-trees. In *ICDE*, pages 258–265, 1996.

[10] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD*, pages 237–246, 1993.

[11] H. Cao, N. Mamoulis, and D. W. Cheung. Discovery of periodic patterns in spatiotemporal sequences. *IEEE Trans. Knowl. Data Eng.*, 19(4):453–467, 2007.

[12] J. Dittrich and B. Seeger. Data redundancy and duplicate detection in spatial join processing. In *ICDE*, pages 535–546, 2000.

[13] A. Eldawy and M. F. Mokbel. SpatialHadoop: A mapreduce framework for spatial data. In *ICDE*, pages 1352–1363, 2015.

[14] A. Eldawy and M. F. Mokbel. The era of big spatial data. *PVLDB*, 10(12):1992–1995, 2017.

[15] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.

[16] R. H. Güting. An introduction to spatial database systems. *VLDB Journal*, 3(4):357–399, 1994.

[17] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[18] E. H. Jacox and H. Samet. Spatial join techniques. *ACM Transactions on Database Systems*, 32(1):7, 2007.

[19] A. Kipf, H. Lang, V. Pandey, R. A. Persa, P. A. Boncz, T. Neumann, and A. Kemper. Adaptive geospatial joins for modern hardware. *CoRR*, abs/1802.09488, 2018.

[20] N. Koudas and K. C. Sevcik. Size separation spatial join. In *SIGMOD*, pages 324–335, 1997.

[21] S. T. Leutenegger, J. M. Edgington, and M. A. López. STR: A simple and efficient algorithm for r-tree packing. In *ICDE*, pages 497–506, 1997.

[22] J. Liagouris, N. Mamoulis, P. Bouros, and M. Terrovitis. An effective encoding scheme for spatial RDF data. *PVLDB*, 7(12):1271–1282, 2014.

[23] M.-L. Lo and C. V. Ravishankar. Spatial hash-joins. In *SIGMOD*, pages 247–258, 1996.

[24] P. A. Longley, M. Goodchild, D. J. Maguire, and D. W. Rhind. *Geographic Information Systems and Science*. Wiley Publishing, 3rd edition, 2010.

[25] N. Mamoulis. Spatial join. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems*, pages 2707–2714. Springer US, 2009.

[26] N. Mamoulis. *Spatial Data Management*. Morgan & Claypool Publishers, 2011.

[27] H. Markram, K. Meier, T. Lippert, S. Grillner, R. S. Frackowiak, S. Dehaene, A. Knoll, H. Sompolinsky, K. Verstreken, J. DeFelipe, S. Grant, J. Changeux, and A. Saria. Introducing the human brain project. In *FET*, pages 39–42, 2011.

[28] M. McKenney, R. Frye, M. Dellamano, K. Anderson, and J. Harris. Multi-core parallelism for plane sweep algorithms as a foundation for GIS operations. *GeoInformatica*, 21(1):151–174, 2017.

[29] S. Nobari, Q. Qu, and C. S. Jensen. In-memory spatial join: The data matters! In *EDBT*, pages 462–465, 2017.

[30] 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.

[31] V. Pandey, A. Kipf, T. Neumann, and A. Kemper. How good are modern spatial analytics systems? *PVLDB*, 11(11):1661–1673, 2018.

[32] J. M. Patel and D. J. DeWitt. Partition based spatial-merge join. In *SIGMOD*, pages 259–270, 1996.

[33] M. Pavlovic, T. Heinis, F. Tauheed, P. Karras, and A. Ailamaki. TRANSFORMERS: robust spatial joins on non-uniform data distributions. In *ICDE*, pages 673–684, 2016.

[34] M. Pavlovic, F. Tauheed, T. Heinis, and A. Ailamaki. GIPSY: joining spatial datasets with contrasting density. In *SSDBM*, 2013.

[35] D. Piatov, S. Helmer, and A. Dignös. An interval join optimized for modern hardware. In *ICDE*, 2016.

[36] S. K. Prasad, M. McDermott, S. Puri, D. Shah, D. Aghajarian, S. Shekhar, and X. Zhou. A vision for gpu-accelerated parallel computation on geo-spatial datasets. *SIGSPATIAL Special*, 6(3):19–26, 2014.

[37] F. P. Preparata and M. I. Shamos. *Computational Geometry - An Introduction*. Springer, 1985.

[38] S. Qi, P. Bouros, and N. Mamoulis. Efficient top-k spatial distance joins. In *Advances in Spatial and Temporal Databases - 13th International Symposium, SSTD 2013, Munich, Germany, August 21-23, 2013. Proceedings*, pages 1–18, 2013.

[39] S. Ray, B. Simion, A. D. Brown, and R. Johnson. Skew-resistant parallel in-memory spatial join. In *SSDBM*, pages 6:1–6:12, 2014.

[40] I. Sabek and M. F. Mokbel. On spatial joins in mapreduce. In *SIGSPATIAL/GIS*, 2017.

[41] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[42] T. Seidl, S. Fries, and B. Boden. MR-DSJ: distance-based self-join for large-scale vector data analysis with mapreduce. In *BTW*, pages 37–56, 2013.

[43] 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.

[44] M. Tang, Y. Yu, Q. M. Malluhi, M. Ouzzani, and W. G. Aref. Locationspark: A distributed in-memory data management system for big spatial data. *PVLDB*, 9(13):1565–1568, 2016.

[45] F. Tauheed, T. Heinis, and A. Ailamaki. Configuring spatial grids for efficient main memory joins. In *BICOD*, 2015.

[46] D. Xie, F. Li, B. Yao, G. Li, Z. Chen, L. Zhou, and M. Guo. Simba: spatial in-memory big data analysis. In *SIGSPATIAL/GIS*, pages 86:1–86:4, 2016.

[47] S. You, J. Zhang, and L. Gruenwald. Large-scale spatial join query processing in cloud. In *CloudDB, ICDE Workshops*, pages 34–41, 2015.

[48] J. Yu and M. Sarwat. Geospatial data management in apache spark: A tutorial. In *ICDE*, pages 2060–2063, 2019.

[49] J. Yu, Z. Zhang, and M. Sarwat. Spatial data management in apache spark: the geospark perspective and beyond. *GeoInformatica*, 23(1):37–78, 2019.

[50] E. T. Zacharatou, H. Doraiswamy, A. Ailamaki, C. T. Silva, and J. Freire. GPU rasterization for real-time spatial aggregation over arbitrary polygons. *PVLDB*, 11(3):352–365, 2017.

[51] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *HotCloud*, 2010.

[52] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao. All-nearest-neighbors queries in spatial databases. In *SSDBM*, pages 297–306, 2004.

[53] S. Zhang, J. Han, Z. Liu, K. Wang, and Z. Xu. SJMR: parallelizing spatial join with mapreduce on clusters. In *CLUSTER*, pages 1–8, 2009.

[54] X. Zhou, D. J. Abel, and D. Truffet. Data partitioning for parallel spatial join processing. In *SSD*, pages 178–196, 1997.

[55] G. Zimbrao and J. M. de Souza. A raster approximation for processing of spatial joins. In *VLDB*, pages 558–569, 1998.