The VLDB Journal (2024) 33:1155–1177
https://doi.org/10.1007/s00778-024-00836-1

SPECIAL ISSUE PAPER

A learning-based framework for spatial join processing: estimation, optimization and tuning

Tin Vu[1] · Alberto Belussi[2] · Sara Migliorini[2] · Ahmed Eldawy[3]

Received: 8 February 2023 / Revised: 26 December 2023 / Accepted: 2 January 2024 / Published online: 13 February 2024

Abstract
The importance and complexity of spatial join operation resulted in the availability of many join algorithms, some of which are tailored for big-data platforms like Hadoop and Spark. The choice among them is not trivial and depends on different factors. This paper proposes the first machine-learning-based framework for spatial join query optimization which can accommodate both the characteristics of spatial datasets and the complexity of the different algorithms. The main challenge is how to develop portable cost models that once trained can be applied to any pair of input datasets, because they are able to extract the important input characteristics, such as data distribution and spatial partitioning, the logic of spatial join algorithms, and the relationship between the two input datasets. The proposed system defines a set of features that can be computed efficiently for the data to catch the intricate aspects of spatial join. Then, it uses these features to train five machine learning models that are used to identify the best spatial join algorithm. The first two are regression models that estimate two important measures of the spatial join performance and they act as the cost model. The third model chooses the best partitioning strategy to use with spatial join. The fourth and fifth models further tune two important parameters, number of partitions and plane-sweep direction, to get the best performance. Experiments on large-scale synthetic and real data show the efficiency of the proposed models over baseline methods.

Keywords Spatial join · Cost model · Cardinality estimation · Machine learning · Query optimization

1 Introduction

In recent years, there has been a substantial surge in the accumulation of extensive spatial data from diverse sources, including satellite imagery [24], social networks [41], smartphones [33], and VGI [31]. To illustrate, users generate an average of 500 million daily tweets [57], reflecting various spatial origins. Additionally, NASA EOSDIS consistently integrates 6.4 TB of data into its repositories daily [55]. The traditional Spatial DBMS technology struggled to handle these petabytes of data, prompting the emergence of numerous significant spatial data management systems, such as SpatialHadoop [17], Hadoop-GIS [2], SparkGIS [6], Beast [16], Apache Sedona (formerly known as GeoSpark) [69], Simba [66], and many others [19]. One of the most important and challenging operations in all these systems is *spatial join*. Apache Spark is widely popular in data analytics due to its efficient processing of large-scale datasets, in-memory computing engine, and extensive libraries for machine learning and graph processing. Spatial join is an integral component of analytical queries on Apache Spark, enabling the combination of datasets based on their spatial relationships. By linking elements from different datasets based on their spatial proximity or containment, spatial join plays a vital role in various applications, such as geographical analysis, urban planning, and location-based services. However, due to the

This work is supported in part by the National Science Foundation (NSF) under Grants IIS-1954644, IIS-1838222 and CNS-1924694 and by "Progetto di Eccellenza" of the Computer Science Department, University of Verona, Italy.

✉ Tin Vu
khactinvu@microsoft.com

Alberto Belussi
alberto.belussi@univr.it

Sara Migliorini
sara.migliorini@univr.it

Ahmed Eldawy
eldawy@ucr.edu

1 Microsoft Corporation, Redmond, USA

2 University of Verona, Verona, Italy

3 University of California, Riverside, USA

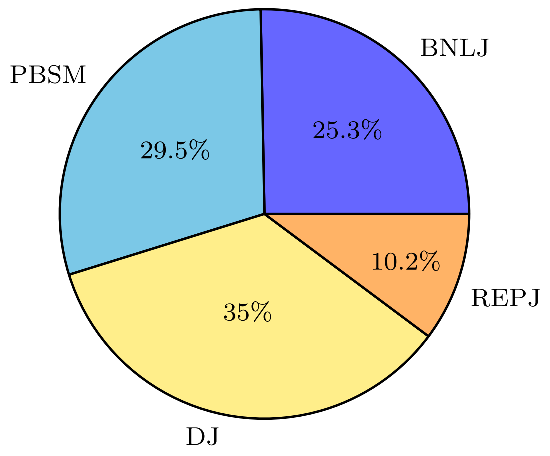

Fig. 1 Distribution of best join algorithm in terms of running time

inherent complexity of spatial data processing, optimizing spatial join operations becomes crucial to ensure efficient query execution and minimize computational overhead. As the volume and complexity of spatial data continue to grow, effective optimization techniques are essential to harness the full potential of spatial join and enhance the overall performance of analytical workflows on Apache Spark. The complexity of the spatial join and the modern big-data systems make these problems extremely difficult. For example, Fig. 1 shows the distribution of the best out of four join algorithms (will be discussed in Sect. 2) in running time when we execute them on a good amount of different datasets [59]. This distribution indicates that it is challenging to choose an appropriate algorithm for a random join input. Our experiment showed that sometimes choosing inappropriate join algorithm could make the join time 5 times slower when compared to the fastest one. For example, given a same join inputs of 128 MB and 2048 MB datasets, the running time of DJ, PBSM, RepJ, BNLJ are 243, 49, 91, 223 s, respectively. This motivated our research to investigate the insights of distributed spatial join algorithms. The experimental results and datasets are publicly available in our Github repository [59]. User can either download the datasets directly or reproduce them using our spatial data generator [35, 65].

Spatial join is one of the most resource demanding operations in spatial databases and it becomes even more challenging with big data [9, 21, 53]. Spatial join is commonly processed using the filter and refinement approach. The filter step only considers the minimum bounding rectangle (MBR) of geometries while the refinement step considers the actual geometry definition. The filter step is the one that involves partitioning and parallelization and this work focuses on that step. In some cases when the complexity of geometries is very high, the refinement step dominates the processing time [29] and needs further attention but this is outside the scope of this paper. Efficient distributed spatial join algorithms work on two main phases, partition and join. The partition step splits the data into smaller parts that can be processed independently. The join step processes each of these partitions individually to produce the final answer.

Due to the complexity of the process, the best algorithm has to balance the computation across machines, reduce disk access from the distributed file system, and minimize network overhead. At the same time, the skewed distribution of the inputs and the hardware specification of the machines have to be taken into account. The complexity of the problem encouraged researchers to develop many spatial join algorithms for big data [18, 34, 68]. Moreover, each algorithm might need to specify some configuration parameters according to the characteristics of the input datasets. These parameters can have a significant impact on the performance of such algorithms. This situation creates a complex query optimization problem to choose the best spatial join algorithm and to tune the configuration parameters for individual algorithms given the input datasets and hardware resources.

Traditionally, this query optimization problem has been addressed using theoretical cost models [3, 7, 27, 28]. With the rise of big-data, some of these cost models have been ported to MapReduce and similar systems [4, 52]. Unfortunately, these theoretical models are limited due to some strong assumptions such as the uniformity of the input datasets or about the query processing engine, e.g., Hadoop MapReduce, which limit their use in practice.

The advancement of machine learning resulted in a new generation of query optimizers that rely on data-driven models for database operations including join [37, 38, 43, 44, 67]. However, these models are limited to equi-join and cannot catch the complex logic of spatial join. Further, most of this work assumes that the same collection of datasets are used for training and testing which limit the applicability of the produced models.

This paper proposes the first *learning-based framework for distributed processing of spatial join on big data*. This cost model can be abstracted as a set of complex functions that estimate specific values, e.g., a configuration parameter, selectivity, or best algorithm, based on some descriptors for the input. The main challenge with this approach is to build a model that balances *generality* and *practicality*. On one side, the model should be general so that it can be ported to different datasets, spatial join algorithms, and systems. On the other side, it needs to be practical by providing a simple output that can be directly used by the query optimizer such as the estimated result size or the best algorithm to run.

To address the challenges above, we propose a machine learning-based framework, called Spatial Join Machine Learning (SJML), that consists of five levels addressing various aspects of query optimization. The first level builds a *cardinality estimation* model that learns the result size independently of the algorithm. The input features used in this level cover individual dataset attributes, e.g., size and some skewness measures, and other features based on the *convoluted histogram*, which catch the joint distribution of the two datasets that is important for spatial join. The second level builds a separate model for each spatial join algorithm that predicts the *number of geometric comparison operations*, which is an algorithm-specific but hardware-independent feature. These first two levels provide quick estimation of result size and computation cost that are helpful for performance evaluation. Then, the third level builds a classification model, which is able to predict the *best partitioning algorithm* that determines if and how each of the input datasets will be partitioned. After that, the fourth level defines a model that estimates the *number of partitions for the partitioning step* in case both inputs will be repartitioned in the PBSM algorithm [47]. Finally, the fifth level defines a model that chooses *how the plane-sweep join algorithm will run in each partition*, i.e., along the x- or y-axis.

In Sect. 7, we validated that the proposed framework can work with spatial datasets in different scales, where the geometry types are point or rectangle. The spatial join predicate used in the paper is intersection join. However, this does not limit the framework to apply to other geometry types or join predicates. For example, any geometry types could be represented by a minimum bounding rectangle (MBR), which can filter the potential join pairs. In addition, other predicates, for example distance join, could be translated into an intersection join. In particular, if we want to find all geometry pairs (g_1, g_2) where the distance is less than d, we can expand g_1 and g_2 by $d/2$ and verify whether the two expanded geometries G_1 and G_2 are intersecting.

This proposed architecture gives system designers the flexibility of choosing the level that matches their application needs from the most general (levels 1 & 2) to the most specific ones (level 3–5). Moreover, it allows us to train some of these models once and share them. For example, the cardinality estimation model in level 1 can be used regardless of the algorithm. Similarly, the models in level 2 are hardware-independent so they can be ported to any hardware. All the proposed models in this paper are independent and could be used either separately or together, depending on the requirements of the considered problem. In addition, the proposed framework could be extended to new spatial join algorithms other than four algorithms we are studying in this paper.

The proposed framework is open source.[1] In particular, we implement the proposed models using *scikit-learn* [49] and train/test them on different datasets. To train the model, we use an open-source spatial data generator [65] to generate hundreds of datasets by varying the data distribution and size. In addition, we train on subsets of publicly available real data from UCR-Star [30]. Together, the synthetic and real data generate a rich training set with thousands of training points. The results show that the proposed model can estimate the cardinality of the spatial join with an error of as low as 8% when compared to the ground truth. It can also predict the number of MBR tests for different algorithms with a reasonable error. Moreover, we tested the end-to-end framework in predicting the best algorithm in terms of running time. Finally, additional tests have been performed to evaluate the accuracy of the prediction for the configuration parameters, which always showed a considerable improvement with respect to the baseline methods.

In summary, we make the following contributions:

1. We design and implement the first machine learning-based model to predict the best algorithm for spatial join in terms of their running times.
2. We describe an end-to-end process for generating datasets, extracting features, training models, and evaluating them for query optimization on spatial join.
3. We break down the algorithm selection model into separate models that predict join selectivity, number of MBR tests, and fastest algorithm.
4. We carry extensive experiments to validate the advantages of the proposed models over the existing solutions.

Notice that, with respect to our previous work [61], this journal paper presents the following additional contributions:

1. We add two new models to the proposed framework. The first one tunes the number of partitions for the PBSM algorithm and the second optimizes the plane-sweep join algorithm that runs locally on each partition. The changes are made at Sect. 6.4 and Sect. 6.5.
2. We introduce new dataset features that help in capturing additional dataset characteristics to help with training the new models. The changes are made at Sect. 5.

This paper aims to address the optimization of the spatial join operation within distributed processing frameworks, specifically focusing on Apache Spark. While other optimization challenges, such as spatial partitioning, have been explored in our previously published works [60, 63, 64], this research centers on enhancing the efficiency and performance of spatial joins.

The rest of this paper is organized as follows. Section 2 illustrates issues and solutions for the execution of spatial join in presence of big input datasets. Section 3 discusses the related work. Section 4 describes the process of our proposed framework for cost model estimation. Section 5 details the training and test process including data generation and preparation. Section 6 describes the five proposed models. Section 7 gives the results of our experiments. Finally, Sect. 8 concludes the papers and discusses future works.

[1] https://github.com/tinvukhac/learned-spatial-join.

2 Background

Spatial join is one of the most important operations in geospatial applications, since it is frequently used for performing data analysis involving geographical information from multiple datasets. With the advent of big data era, the size of the datasets to be joined is considerably increased. This made the spatial join operation one of the most time-consuming operation to be performed, leading to the definition of several different algorithms in order to optimize its execution.

From one side, these techniques try to capture and exploit the characteristics of the two datasets at hand in order to identify the best way to join them. Such characteristics includes both the peculiar features of each dataset alone (e.g., number and size of geometries, their spatial distribution, and so on), and the combined features of the two datasets together (e.g., their overlapping and relative displacement). From the other side, when the join operation is performed in a MapReduce environment, it induces additional complexity since it requires to consider two datasets simultaneously, while systems like Hadoop and Spark have been tailored for processing only one argument at time.

At a high-level, big-data spatial join algorithms run in two phases, partitioning and join. In the partitioning phase, the two inputs are partitioned into small partitions that can be processed independently. The join phase processes these partitions in parallel to produce the final results as a set of record pairs. The various spatial join algorithms differ in how they partition the input and join each partition. Some algorithms can only work if one or both inputs are already partitioned, i.e., indexed. Figure 2 summarizes the join strategy of four different join algorithms that will be considered in this paper. Advantages and disadvantages of the analyzed algorithms are listed in Table 1.

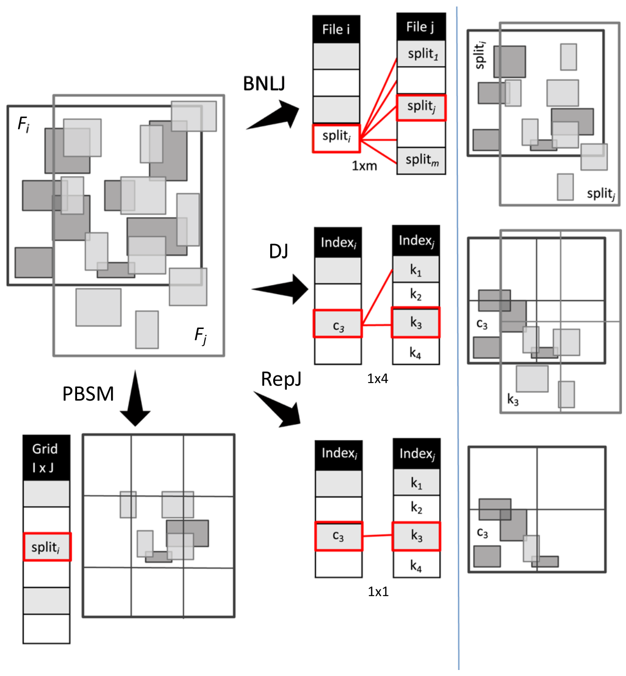

Fig. 2 The join strategy of different join algorithms

The simplest spatial join algorithm is represented by the Block Nested Loop Join (BNLJ)[9]. It is a map-only job, i.e., has no reducers, and clearly it is classified as a map-side join. It works on non-indexed data and the input for the mappers is prepared by a reader, which generates a "combined split" for each possible pair of input splits coming from the two datasets. In other words, given two input files F_i and F_j, a combined split is produced for each pair of splits belonging to the cross product $F_i \times F_j$. Inside the BNLJ, each mapper loads the content of its combined split into two lists, then it applies the plane sweep algorithm for checking the intersection between the geometries in the two lists. Our previous work [9] provides a comprehensive cost model for distributed spatial join algorithms. It showed the cases where BNLJ outperformed other spatial join algorithms. One case where BNLJ is superior is when one dataset is small, e.g., one or a few HDFS blocks. In this case, BNLJ behaves like a broadcast join which broadcasts the small dataset and partitions the big dataset. Other algorithms will behave similarly but they will pay the overhead of spatial partitioning which is not very effective in this case. In general, BNLJ is superior when the overhead of partitioning over weights its benefits.

The main limitation of BNLJ is that it does not consider the spatial characteristics of datasets at hand, in particular each split could contain data residing in very different locations, increasing the number of comparisons to be performed and the number of mappers to be instantiated. For this reason a first enhancement of this algorithm is represented by the Distributed Join with Index (DJ) [18, 68], a MapReduce adaptation of the Grid File Spatial Join algorithm [36]. This variant is very similar to BNLJ and it is again a map only job (and consequently a map-side join). However, in this case the reader works on indexed data, namely each input dataset has been preliminary partitioned into splits containing only nearby objects. In this way the reader is able to produce a limited number of combined splits w.r.t. the cross product, namely it will produce only the pair of cells (splits) with non-empty spatial overlap. Clearly this solution reduces the number of instantiated mappers, but at the same time each of them has potentially more work to do w.r.t. the ones instantiated by BNLJ (i.e., mappers would potentially produce in average a greater number of result geometries). Moreover, the preliminary indexing or partitioning activity comes with its own cost and in some cases it is justified only if such index can be re-used for other operations or analysis. Therefore, the final choice between the two algorithms greatly depends on the single and combined characteristics of the two input datasets.

A problem of the DJ algorithm could be that the two input datasets are typically partitioned by using the best individual index, but they can be very different in terms of grid (i.e., number of cells and shapes of regions covered by each cell), leading to cases in which the benefits induced by the use of such partitions are lost. Therefore, a variant of DJ is represented by the Repartition Join algorithm (REPJ) [18, 68] which is a MapReduce adaptation of the Bulk-Index Join [14]. In this case, instead of using two completely different partitioning grids, one of the two datasets (typically the smaller one) is partitioned w.r.t. the index of the other one. In this way the number of generated combined splits, and consequently the number of mappers, is further reduced and it is a subset of the number of index cells. Also in this case the trade-offs of a preliminary repartition have to be carefully evaluated in each case.

An example of reduce-side spatial join is represented by the Spatial Join Map Reduce (SJMR) [70] which is a MapReduce implementation of the Partition Based Spatial Merge Join (PBSM) [48]. This algorithm has been designed to efficiently perform a spatial join on non-indexed datasets. During the map phase the datasets are partitioned w.r.t. a common grid computed on the basis of the joint characteristics of the two datasets, while multiple parallel reducers are responsible for computing the spatial join inside each produced cell.

An important factor to be considered when the spatial join is implemented in a MapReduce environment is the balancing of the work performed by the mappers (or reducers) for testing the join condition. Indeed, it is crucial that there is no worker doing the majority of the work, while others have nothing to do, since in this case the benefits of a parallel execution are completely lost. As regards to the above algorithms while SJMR and REPJ work well when the two datasets are uniformly distributed or share a very similar distribution, the only one that tries to deal with skewness aspect is DJ, since each dataset is partitioned with the best possible technique producing balanced splits. Anyway, even if these splits are individually balanced, they would produce very unbalanced situations when they are combined together.

In this paper, we build our models based on the data collected from four fundamental spatial join algorithms mentioned above: block nested-loop join (BNLJ), partition based spatial merge join (PBSM), distributed index-based join (DJ), and repartition join (REPJ). However, it should be trivial to extend these models with other spatial join algorithms. This is similar to adding a new class to a classification model.

The balancing criteria is usually based on the evaluation of the number of objects contained in each split. However, this can be a simplistic criteria, in particular when we consider the work to be done for actually evaluating a spatial predicate between large or complex geometries (instead of only on their MBRs). Moreover, when large geometries are partitioned, they are typically replicated in any overlapping split, dramatically increasing the amount of work to be done especially when they are not only large, but also described by a great number of points. In order to overcome this situation in [51] the authors proposed a parallel in-memory spatial join, called SPINOJA, which is based on an innovative partitioning technique. This technique is called MOD-Quadtree (metric-based object decomposition quadtree) and is a region quadtree variant that recursively decomposes a split into four equal-sized splits by using a criteria which is not based on the number of objects, but rather on the amount of computation estimated by suitable work metrics. Such metrics include for instance the size of the objects and their complexity (i.e., number of vertices). Moreover, any time a cell decomposition is done, any object that overlaps a newly created (smaller) cell is also decomposed along its boundaries, avoiding expensive replications.

An extension of the traditional spatial join operation is represented by the multi-way spatial join, namely a spatial join where the involved datasets can originate from more than two sources. An example of multi-way spatial join is represented by the query "finding all the forests crossed by a river in each state" which requires to join three different datasets: forests, rivers and states. A straightforward way to generalize a spatial join into a multi-way spatial join is to transform the latter into a set of cascaded pairwise joins. In this case, each pair of datasets could be joined by using one of the above algorithms and then the partial results could be transferred and used as input for the following join operation. However, this solution can lead to a very large amount of communication among cluster nodes. Some optimization are possible for solving this problem, for instance if the implementation is performed in Spark, the intermediate results could be cached in memory [15]. In general, sophisticated techniques could also be defined and studied, as the one in [32], which try to minimize the communication by exploiting the spatial locations of the geometries in the intermediate results. For instance, the intersection between the cell produced by the join and the cells in the next dataset to be joined.

3 Related work

Non-spatial join optimization Due to its popularity and importance, there has been a large body of work for optimizing non-spatial equi-join. There are mainly three problems related to query optimization, *selectivity estimation*, *join cost estimation*, and *join ordering*. Traditionally, theoretical models were proposed to solve these three problems [25, 26, 39, 40, 45, 58]. One of the key challenges is the correlation between join attributes. With the rise of *machine learning*, it was utilized to build sophisticated data-driven models for selectivity estimation [37, 67], join cost estimation [38, 42, 44, 46], and join order enumeration [43]. These ML-based

Table 1 The advantages and disadvantages of different join algorithms

Join algorithm	Advantages	Disadvantages
BNLJ	Simple; works on non-indexed data	Is not spatial-awareness
DJ	Spatial-awareness; less # of mappers	Both inputs must be indexed; potentially more works for each join pair
RepJ	Less # of mappers	One input must be indexed
PBSM	Works on non-indexed data	The partitioning process is costly

methods suffer from two limitations. First, they only work with equi-joins and do not support the complex logic of spatial join. Second, existing models are trained on a small number of tables and can only work with these tables. For example, existing techniques model the input table using a one-hot vector that defines which table to work with.

Spatial join optimization Given the high cost of spatial join, its optimization also was rigorously studied through the two problems of selectivity estimation and join cost estimation. Effective formulas have been proposed for uniformly distributed datasets in [4], but their extension to skewed distributions were not straightforward. For accurate selectivity estimation, a method was proposed that computes the correlation fractal dimension of the considered spatial datasets and applying a power law for self-join [7] and binary join [27]. However, these methods were limited to point datasets with distance join predicate.

To optimize distributed spatial join queries, a cost-based and rule-based query optimizer was proposed for MapReduce [52]. It breaks down the spatial join into two phases, partition and join, and decides which technique to use in each phase. This work has two limitations. First, the cost-based model requires the measurement of eight parameters to catch the characteristics of the hardware, algorithms, and data. Second, it did not consider all join algorithms, e.g., block nested loop join. A more detailed model was proposed in [9] which breaks down the cost into CPU, local and network I/O components. It overcomes the limitations of the earlier work as it uses only simple data statistics and supports more algorithms but it is limited to uniformly distributed data.

Dataset's histogram has proved its popularity in database systems for the selectivity estimation problem due to its efficiency in both computation cost and required storage space [1, 54]. In order to give the machine learning model a global and local picture of how two join input datasets overlap with each other, the convoluted histogram of two datasets should be provided. The paper [1] proposes the min-skew algorithm to compute the histogram of a single dataset, which can be used to combine two datasets histogram if they are aligned to each other. The recent work [53] proposed an efficient method to merge two non-aligned histograms, which is more flexible for a wider range of spatial datasets.

This paper proposes the first ML-based model for distributed spatial join for cost estimation and selectivity estimation in both partitioning phase and join phase. It differs from existing ML-based query optimizers as it supports spatial join and can apply to any input data that was not part of the training process. It also overcomes the limitations of existing theoretical spatial join models as it supports skewed data including real datasets and it works on well-defined data statistics that can be computed with a simple data scan. Papers related to this work are summarized in Table 2.

4 Overview of SJML

This section gives an overview of the proposed Spatial Join Machine Learning (SJML) framework illustrated in Fig. 3. SJML contains five machine learning models (M1-M5) that assist in all stages of running a spatial join query. Initially, before starting the actual join operation, SJML provides some estimated statistics on the performance of the query. More specifically, M1 estimates the result size which is an algorithm-independent metric and can be used for cardinality estimation of complex queries. M2 estimates the computation cost of each algorithm in terms of number of rectangle overlap tests, a hardware-independent metric, which can be very useful for cost estimation. Since these estimates are provided before spatial join starts, they do not act on the actual data, but only on some statistics extracted from the data.

Table 2 Related work in join optimization

Join type	Non-spatial	Spatial
Problems	Selectivity estimation, join cost, and join ordering	Selectivity estimation and join cost estimation
Theoretical	[25, 26, 39, 40, 45, 58]	[3, 4, 7, 27, 28, 52]
ML	[37, 38, 42–44, 46, 67]	This work

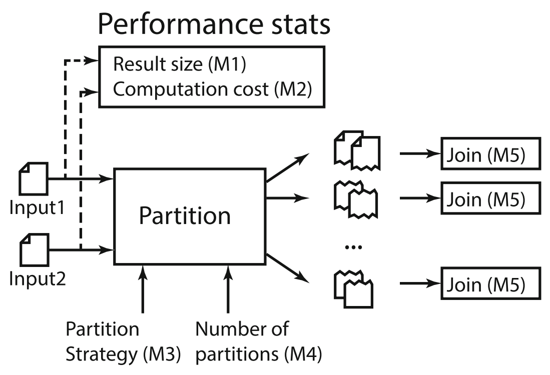

Fig. 3 Overview of SJML

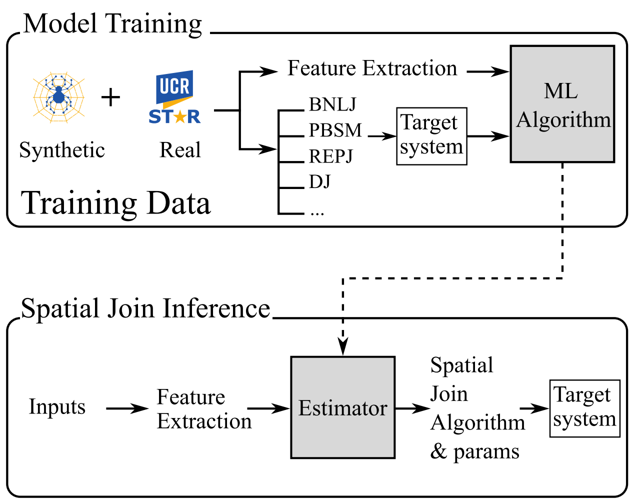

Fig. 4 Training and Inference Process in SJML

The spatial join process itself starts by partitioning the data into smaller partitions that can be processed in parallel. SJML provides M3 that chooses the most efficient partition strategy for the given input. For some partition strategies, e.g., uniform grid partitioning used in PBSM [47], the number of partitions also needs to be tuned, hence, we propose M4 to tune this parameter. Finally, when it comes to the join phase where partitions are processed in parallel, SJML provide M5 that determines the most efficient order of processing each individual partition. In this paper, we use the planesweep join algorithm and M5 decides whether to scan the geometries along the x-axis or y-axis.

SJML is a supervised model that requires a training phase. Most existing ML-based query optimization models can only be applied to the same data distributions it was trained on. However, SJML breaks from this restriction as it builds a dataset-independent model that can be applied to any dataset. In order to obtain this generality, in the training phase, we use Spider [65], a standard spatial data generator, to generate various datasets with diverse characteristics. We further enrich the training set with real data extracted from UCR-Star. The training and inference process is illustrated in Fig. 4. For each pair of datasets, we extract a set of features that the machine-learning models can train on. We also run all available algorithms on each pair of datasets to measure their behavior with such pair of datasets. Each combination results in a training point that can be fed to the proposed ML algorithm to learn how the existing algorithms behave for a specific pair of input datasets. A similar training is done for learning the effect of number of partitions (M4) and processing order (M5). In particular, for M4 we run the spatial join by varying the number of partitions, while for M5 we process each partition along both the x and y axes to learn their effect on performance. Also in this case the training is done by first extracting the same set of features we used before, or by adding some new ones very similar to them.

This generic model can be easily extended to more spatial join algorithms or data processing systems. In this paper, we focus on four fundamental spatial join algorithms, namely, block nested-loop join (BNLJ), partition based spatial merge join (PBSM), distributed index-based join (DJ), and repartition join (REPJ). All these algorithms are implemented in both Hadoop and Spark.

After the models have been built and trained, we can use M1 and M2, if needed, to estimate the result size and processing cost. Then, we use M3 to decide the best partition strategy. If the partition strategy requires the number of partitions to be set, then we use M4 to tune this parameter. Finally, after the data is partitioned, each machine runs M5 independently on each partition to decide how to process it locally. In other words, each partition will have its own decision for processing.

Data Generation To obtain a portable model, we need to make sure that we generate diverse and representative data. We generate data using five different distributions and vary the data distribution parameters such as skewness and geometry size. We also create *compound* dataset that combine two or more simple datasets.

Feature Extraction The most challenging step in the proposed model is the feature extraction step. We need to extract a set of features that are relatively easy to compute and are useful for the spatial join problem. We extract three sets of features. First, we collect statistical-based features such as the cardinality of the input and average geometry area. Second, we collect histogram-based features that represent data distribution and skewness such as box counting. We also introduce a *convoluted histogram* that combines the two datasets together to represent the relationship between them. Finally, we compute partitioning-based features that represent how the data is partitioned and indexed.

5 Model training and testing

The training process runs a supervised training phase to build an ensemble of models able to estimate various cost components of spatial join. First, during this phase a feature extraction step is performed that, given two generic spatial datasets, converts them into a fixed-size feature vector and computes a set of performance metrics. Each model learns a function that maps a subset of the feature vector to one of the performance metrics. Second, to train a generic and representative model, we use Spider [65] to generate diverse synthetic datasets, and UCR-Star [30] to extract real datasets. Finally, we prepare the master datasets that we use for training and testing each of the models that we propose in the paper. The details of these steps are described below.

5.1 Feature extraction

The spatial datasets processed by a spatial join operation are not directly usable by the machine learning models that we propose, since these models typically expect a fixed-size feature vector not a variable size dataset. Therefore, a preliminary step is required that extracts a fixed set of standard features for each pair of datasets D_i and D_j to prepare the data points for model training. All the features that we propose to use can be extracted in a single parallel scan over the data.

Table 3 summarizes all features we use in this paper while the details of their computation are given in the following sections. More specifically, we divide the set of features into two groups, *single* and *combined* dataset statistics. Single dataset statistics are features that are extracted once for each individual dataset separately. Combined statistics represent features that are computed for each specific pair of datasets together. Finally, we collect a set of *metrics* from the execution of each spatial join algorithm that represent the cost of this operation.

5.1.1 Single dataset statistics

The following statistics can be computed individually for each dataset D_* and they do not depend on the join query, since they describe the dataset itself. We divide them into four groups. The first two groups, denoted as P_{size} and P_{dens}, contain a set of associative and commutative functions that can be computed efficiently in one parallel scan. The third group, denoted as P_{dist}, is computed starting from a histogram over the data which can be built in an exact way by using an additional scan, or in an approximated way during the same scan used for the first two groups [53]. The fourth group, denoted as P_{part}, is computed from the partitioning information and does not require to scan the actual records. Further details are provided in the following four definitions.

Definition 1 (*Data size statistics—P_{size}*) Given a dataset D_*, we define:

- $\#geo(D_*)$: number of geometries (records) in D_*, it corresponds to $|D_*|$.
- $size(D_*)$: size of the dataset in bytes. It is computed as $size(D_*) = \sum_{r \in D_*} size(r)$, where $size(r)$ is the size of a record in bytes. This is the size of the entire geometry, not the size of its MBR.

This group of statistics characterize the dimension of the dataset from both a more traditional point of view (i.e., the size in bytes) and a spatial point of view (i.e., number of geometries or number of points).

Definition 2 (*Data density statistics—P_{dens}*) Given a dataset D_*, we define:

- $\mathtt{MBR}(D_*)$: Minimum Bounding Rectangle of D_*. It is defined as $\mathtt{MBR}(D_*) = (\min x, \min y, \max x, \max y)$.
- $mbrArea^{avg}(D_*)$: given the MBR of all geometries r in D_* ($mbr(r)$), it represents the average area of such MBRs: $mbrArea^{avg}(D_*) = \frac{\sum_{r \in D_*} area(\mathtt{MBR}(r))}{|D_*|}$
- $len_x^{avg}(D_*)$ and $len_y^{avg}(D_*)$: given the MBR of all geometries in a dataset, they represent the average length on the x- and y-axis of such MBRs. $len_x^{avg}(D_*) = \frac{\sum_{r \in D_*} len_x(\mathtt{MBR}(r))}{|D_*|}$, while $len_y^{avg}(D_*)$ is calculated similarly.

This group of statistics characterize the extension and the shape of the geometries contained in the dataset from both an absolute and relative point of view.

After the above statistics have been calculated, the histogram of each input dataset is computed. We always create a high-resolution histogram of size 8192×8192. The usefulness of this histogram is twofold: first of all, we use it to apply the box-counting technique and compute two parameters, called E_0 and E_2, which synthetically describe the distribution of the input dataset [10]. Second, it is used to calculate the lower-resolution histograms $H(D_*)$ that will be used for the computation of the convoluted histogram described shortly.

Definition 3 (*Histogram-based statistics—P_{dist}*) Given a dataset D_*, its high-resolution histogram h is generated and some metrics describing the distribution of the geometries in the reference space of D_*, are computed as proposed in [8, 10, 60]. In particular, two numeric values, called E_0 and E_2, derived from the box-counting ($BC^q_{D_*}(r)$) technique [7] are considered. The following formula shows how to compute the function BC on the histogram h with scale r. E_0 and E_2 are the slope of the plot of $BC^0_{D_*}(r)$ and $BC^2_{D_*}(r)$ in log

Table 3 Summary of features

	Group	Feature	Description
Single	P_{size}	$\#geo$	Number of geometries
		$size$	Total input size
	P_{dens}	MBR	Minimum bounding rectangle
		$mbrArea^{avg}$	Average record area
		len_x^{avg}	Average record width
		len_y^{avg}	Average record height
	P_{dist}	E_0, E_2	Box counting with base 0 and 2
	P_{part}	$\#cells$	Number of cells (partitions)
		$\#splits$	Number of splits (blocks)
		TT_{area}	Sum of partition areas
		TT_{margin}	Sum of partition semi-perimeters
		$TT_{overlap}$	Sum of overlap areas between pairs of partitions
		LB	Load balancing: Standard deviation of sizes
		BU	Block utilization: Percentage of block usage
Combined	P_{mbr}	$Area_i$, $Area_j$	Percentage of overlap area occupied by dataset i or j
		$Area_{ix}$	Dataset i overlap percentage on x-axis
		$Area_{jx}$	Dataset j overlap percentage on x-axis
		$Area_{iy}$	Dataset i overlap percentage on y-axis
		$Area_{jy}$	Dataset j overlap percentage on y-axis
		$JS_{i,j}$	Jaccard similarity between the two MBRs
		$JS_{i,j,x}$	Jaccard similarity between the two MBRs on x-axis
		$JS_{i,j,y}$	Jaccard similarity between the two MBRs on y-axis
	P_{ch}	e_0, e_2	Box counting for the convoluted histogram with base 0 and 2

scale, respectively.

$$BC^q_{D_*}(r) = \sum_i h_i(D_*, r)^q$$

where $h_i(D_*, r) = count$(geometries of D_* intersecting the i cell of h with scale r).

Since every dataset has to be partitioned when used in Hadoop and Spark, we devise a fourth set of features that represent the spatial characteristics of this partitioning. Spatial partitioning is characterized by the MBR, the number of geometries, and the total size of each partition. If the data is not spatially partitioned, we assume that the MBR of all partitions is the same as the dataset MBR and that the number of geometries and size is equally split among partitions.

Definition 4 (*Partition statistics—P_{part}*) Given a dataset D_* that has been partitioned with some technique, we identify the following parameters:

- $\#cells(D_*)$: number of cells in which the dataset D_* has been partitioned.
- $\#splits(D_*)$: number of physical blocks containing records of D_*. In case of a uniform distributed dataset, $\#splits(D_*)$ typically coincide with $\#cells(D_*)$. Conversely, in presence of a skewed dataset, there can be empty cells, namely cells corresponding to zero blocks, or overloaded cells, namely cells whose content is further subdivided into many physical blocks.
- Total area of D_*: sum of the area of all partitions.

$$TT_{area}(D_*) = \sum_{c_i \in \mathcal{I}(D_*)} area(c_i) \times c_i.blocks$$

where $\mathcal{I}(D_*)$ is the set of cells used for partitioning the reference space of D_* and $c_i.blocks$ is the number of blocks stored in cell c_i.
- Total margin of D_*: sum of the length of the semiperimeter of all partitions.

$$TT_{margin}(D_*) = \sum_{c_i \in \mathcal{I}(D_*)} sp(c_i) \times c_i.blocks$$

where $sp(c_i)$ is the semiperimenter of the cell c_i, i.e., the sum of its width and height.
- Total overlapping of D_*: sum of the area of the overlapping regions produced by intersecting each cell c_i with

all other cells.

$$TT_{overlap}(D_*) = \sum_{c_i,c_j \in \mathcal{I}(D_*), i \neq j} area(c_i \cap c_j) \times B(c_i, c_j) + \sum_{c_i \in \mathcal{I}(D_*)} area(c_i) \frac{c_i.blocks \times (c_i.blocks - 1)}{2}$$

where $B(c_i, c_j) = c_i.blocks \times c_j.blocks$

- Load balancing: standard deviation of index cells cardinality

$$LB(D_*) = \sqrt{\frac{\sum_{c_i \in \mathcal{I}(D_*)} (c_i.card - avgCard)^2}{|\mathcal{I}(D_*)|}}$$

where $avgCard$ is the average cardinality of the cells belonging to the index $\mathcal{I}(D_*)$.

- Block utilization: average percentage of block usage:

$$BU(D_*) = \frac{\sum_{b_i \in \mathcal{I}(D_*).blocks} (b_i.size/blockSize)}{|\mathcal{I}(D_*).blocks|}$$

These metrics can be easily calculated from the set of metadata describing the partitioning. This set is typically very small and can be processed by a single machine. For example, for disk-based partitioned data, this information is stored in a single master file [4].

5.1.2 Combined statistics

Since spatial join is a binary operation that takes two input datasets, we include additional *combined statistics* that capture the relationship between the two inputs. Therefore, they need to be computed for each pair of datasets that are input of a join query. We further subdivided them into two groups: the first one, denoted as P_{mbr}, describes the degree and kind of overlap between the two input datasets, while the second one, called P_{ch} represents the local properties of such intersection in terms of overlapping geometries.

The first group is described below and it does not require an additional scan over the data, since it can be derived from the summaries previously computed for each dataset individually.

Definition 5 (*MBR overlapping statistics—P_{mbr}*) Given a pair of datasets (D_i, D_j), the following statistics describing their overlapping are computed. The *area, x, y* functions return area, width and height of the geometry, respectively.

- Percentage of the area of D_i occupied by the area of the datasets intersection:

$$Area_i = \frac{area(\texttt{MBR}(D_i) \cap \texttt{MBR}(D_j))}{area(\texttt{MBR}(D_i))}$$

- Percentage of the area of D_j occupied by the area of the datasets intersection:

$$Area_j = \frac{area(\texttt{MBR}(D_i) \cap \texttt{MBR}(D_j))}{area(\texttt{MBR}(D_j))}$$

- Percentage of the width of D_i occupied by the width of the datasets intersection:

$$Area_{ix} = \frac{x(\texttt{MBR}(D_i) \cap \texttt{MBR}(D_j))}{x(\texttt{MBR}(D_i))}$$

- Percentage of the width of D_j occupied by the width of the datasets intersection:

$$Area_{jx} = \frac{x(\texttt{MBR}(D_i) \cap \texttt{MBR}(D_j))}{x(\texttt{MBR}(D_j))}$$

- Percentage of the height of D_i occupied by the height of the datasets intersection:

$$Area_{iy} = \frac{y(\texttt{MBR}(D_i) \cap \texttt{MBR}(D_j))}{y(\texttt{MBR}(D_i))}$$

- Percentage of the height of D_j occupied by the height of the datasets intersection:

$$Area_{jy} = \frac{y(\texttt{MBR}(D_i) \cap \texttt{MBR}(D_j))}{y(\texttt{MBR}(D_j))}$$

- Jaccard similarity:

$$JS_{i,j} = \frac{area(\texttt{MBR}(D_i) \cap \texttt{MBR}(D_j))}{area(\texttt{MBR}(D_i) \cup \texttt{MBR}(D_j))}$$

- Jaccard similarity on x-axis:

$$JS_{i,j,x} = \frac{x(\texttt{MBR}(D_i) \cap \texttt{MBR}(D_j))}{x(\texttt{MBR}(D_i) \cup \texttt{MBR}(D_j))}$$

- Jaccard similarity on y-axis:

$$JS_{i,j,y} = \frac{y(\texttt{MBR}(D_i) \cap \texttt{MBR}(D_j))}{y(\texttt{MBR}(D_i) \cup \texttt{MBR}(D_j))}$$

In order to combine the information stored in each high-resolution histogram previously computed separately on the datasets D_i and D_j, we introduce the concept of *convoluted histogram*. A convoluted histogram is simply the overlay of the individual histograms from the two datasets, one on top of the other. The goal of this histogram is to give to the machine learning model a local picture of how the two datasets overlap with each other, which is extremely important for estimating the cost of the spatial join query.

The computation of a *convoluted histogram* would require an additional scan over the two datasets together. Indeed, to compute a *simple* histogram for one dataset, we overlay a uniform grid and count the number of records in each grid cell. If a record overlaps multiple cells, we count it only in the grid cell that contains its centroid to avoid overcounting. Finally, we normalize the histogram by dividing all counts by the largest number to prepare it for machine learning processing. Clearly, given two generic datasets D_i and D_j, even if the grids used for computing their simple histograms is the same in terms of number of cells, they can differ in terms of placement and cell dimension. However, to compute a *convoluted* histogram for two datasets, we need to apply the same grid for both of them, whose dimension covers the space occupied by the two inputs together. This ensures that their relative positions in space is taken into account in the convoluted histogram. To do that, we define the grid based on the minimum bounding rectangle (MBR) of the *union* of the two datasets, i.e., the enlarged MBR that covers both datasets. Since the convoluted histogram requires knowledge of the two datasets before it is computed, it is not possible to precompute before the join query runs, e.g., as a part of data preprocessing and indexing. On the other hand, computing it on the fly when the join query comes would further delay the query and would defy the whole purpose of the spatial join query optimization problem.

To efficiently compute the convoluted histogram, we do it in two steps. The first step, which can be done offline, computes a simple histogram for each datasets based on its own MBR. Then, when the join query comes, we define the grid of the convoluted histogram based on the union MBR of the two datasets. After that, we *transform* the two simple histograms to the new grid to define the convoluted histogram as explained in [53]. The key idea of this transformation is to run a sort-merge-like algorithm that maps each bin in one of the two histograms to the corresponding bin in the convoluted histogram. In addition, it defines a multiplier ratio that is computed based on the amount of spatial overlap between the source and target bins in the two histograms. Note that this transformation is approximate and assumes that the data in each bin is uniformly distributed but it is sufficient for our purpose. Further details about this operation is explained in [53]. The paper [53] also reported its performance in computing the histogram for datasets in different scales. In particular, it takes a few seconds to compute the histogram of a dataset with size of few hundreds megabytes. Meanwhile, the total join time for two datasets with size of few hundreds megabytes could be 300 s as shown in Fig. 6. In a larger scale, it could take up to 100 s to compute the histogram of datasets with size from 5.5 GB to 168 GB. But the joining time for these datasets could be several hours. Therefore, the computation cost of the convoluted histogram is relatively small when compared to the total running time of the join operation. This allows us to extract the dataset insights with a reasonable overhead.

Definition 6 (*Convoluted histogram-based statistics—P_{ch}*) Given a pair of datasets (D_i, D_j), some metrics describing their mutual distribution and based on the box-counting technique are computed. In particular, E_0 and E_2 are calculated starting from the convoluted histogram of D_i and D_j as shown in Def. 3. They are able to describe the distribution and density of the overlapping areas between D_i and D_j. For sake of clarity we call them e_0 and e_2, leaving E_0 and E_2 for single histograms.

This collection of parameters will be used for setting the machine learning model and, in particular, they will be used for generating the input vector of the model.

5.1.3 Performance metrics

The following metrics are collected from the result of spatial join execution on each pair of datasets. The proposed models are trained on these collected metrics and try to estimate them during the inference step, as further detailed in Sect. 4.

- *Join selectivity* the first performance metric is the *join selectivity* (σ_{JN}), namely the cardinality of the spatial join divided by the cardinality of the cross product of the two datasets. It is computed as:

$$\sigma_{JN}(D_i, D_j) = \frac{|D_i \bowtie D_j|}{|D_i| \cdot |D_j|}$$

- *MBR test selectivity* the second performance metric is the *MBR test selectivity* (σ_{MT}), computed as:

$$\sigma_{MT}(D_i, D_j, A_i) = \frac{MC(D_i, D_j, A_i)}{|D_i| \cdot |D_j|}$$

  i.e., the number of MBR tests (or MBR count MC) that each specific join algorithm A_i requires divided by the cardinality of the cross product of the two datasets. This is an important machine-independent metric that is strongly correlated with the running time.
- *Join running time* the third metric is the running time of a join algorithm on a specific hardware, e.g., a cluster of machines.
- *Best join algorithm* the fourth performance metric determines the best algorithm in terms of running time, i.e., one of the four labels: BNLJ, PBSM, DJ, and REPJ.

5.2 Training data generation

In this step, we produce the training data that will be used to build the SJML model. Our goal is to produce a universal model that can work with any pair of input datasets, so it is vital that the training data is diverse and representative of a large swath of distributions. At the same time, since the data generation process includes the effective execution of the spatial join on the selected pairs of datasets, we need to ensure that this process will not take too long. We can summarize the goals of this steps as follows:

1. Generate representative data that can train an accurate and universal model.
2. Reduce the overhead of generating the data points, in particular the spatial join execution.
3. Produce data with the ability to train all the proposed models, i.e., join selectivity, MBR test selectivity, and best algorithm.

To generate representative data, we combine synthetic data from the Spider [35, 65] data generator, and publicly available real data from UCR-Star [30]. For synthetic data, we generate datasets of five different distributions provided by Spider, namely, uniform, parcel, bit, Gaussian, and diagonal. Additionally, we apply various transformations to the generated datasets such as translation, scaling, and rotation to ensure the coverage as much cases as possible, such as the ones in which the input datasets are not perfectly aligned. All transformations are represented using a single affine transformation matrix, which allows us to control all these transformations and ensure we have diverse distributions. Moreover, we combine some of these generated datasets to produce complex distributions, e.g., diagonal and Gaussian together. In total, we obtain 318 synthetic datasets. This allows us to produce more than 10,000 data points after running the join operation on these datasets.

In addition to synthetic data, we also use real datasets from the public repository, UCR-Star. Since the number of real datasets is relatively limited and they are harder to retrieve and process, we generate arbitrarily many real datasets by extracting subsets of the real data with random search windows. This ensures that the distributions of the real data are diverse and representative of many real situations. We chose the OSM buildings, lakes, and roads datasets, and US Census linearwater, edges, and faces datasets. In total, we have 94 real datasets that range from 134 MB (few HDFS blocks) to 7 GB (many HDFS blocks). This variation of number of blocks will result in the high variation of choice of the best join algorithm.

To reduce the overhead of generating the data points, we generate input datasets at three scales: small, medium, and large. These are at the order of 1–2 MB, 10–20 MB, and 1–2 GB, respectively. The key idea is that smaller datasets are easier to generate and process which allow us to generate thousands of training points in a very short time. However, small datasets might not hold a good representation of performance characteristics, which can only be tested with larger data. Thus, we generate the medium and larger datasets which can better catch performance behavior of spatial join algorithms as described shortly. Table 4 summarizes the datasets being used in our experiments.

To be able to train all models, we divide the training data among the three models as follows. First, all the datasets are used to train the model estimating the join selectivity, since this metric is not affected by the data size, but only by the relation between the datasets. In other words, if the dataset gets bigger but all data characteristics remain the same, the selectivity will not change. To further clarify that to the reader, we use Spider to generate four groups of datasets. For each group, we fix the dataset characteristics and increase the data size from 1 MB to 50 MB. For each dataset size, we run spatial join and measure the join selectivity as the result cardinality divided by the product of the two input cardinalities. As shown in Fig. 5a, the selectivity of spatial join is almost a constant which confirms this point. At the same time, as Fig. 5b shows, joining the 1 MB dataset (*Small scale dataset* in the figure) is about 10 times faster than joining the 50 MB datasets (*Large scale dataset* in the figure). Hence, we can use this method to generate thousands of training points in a reasonable time without sacrificing the quality of the model.

Second, to train the model for estimating the MBR test selectivity, we use only medium and large datasets. While MBR selectivity might not be affected by cardinality, the way we partition the data is. Therefore, when using the medium dataset (scale of 10–20 MB), we adjust the partition size to be 128 KB. This results in a partitioning that has similar characteristics to partitioning 10–20 GB dataset with a partition size of 128 MB.

Third, to train the model for fastest algorithm detection, we use the large datasets which show the actual performance of the machines on big data. We use datasets that are big enough to utilize all executor nodes in the cluster. The goal is to avoid the case where only a few executor nodes are working, since they do not represent the real performance of the cluster. This is similar to focusing on asymptotic behavior of algorithms when running on large enough input data. Overall, our proposed framework focus on HDFS-based data systems (Hadoop, Spark) and give more priority for problems on large datasets.

5.3 Training set preparation

In our system, we organize the training data in a tabular form. Each row of the training data includes a list of features and the output metric. The metric could be the join selectivity, the

Table 4 The datasets to train and test the proposed models

Datasets	Scale	Distribution	# of geometries	Size	# of datasets	# join pairs
D1	Small	Uni, Dia, Gaus, Par, Bit	10,000–20,000	1–2 MB	160	7,140 ($D1 \bowtie D1$)
D2	Medium	Uni, Dia, Gaus, Par, Bit	100,000–200,000	10–20 MB	50	5,340 ($D2 \bowtie D3$)
D3	Large	Uni, Dia, Gaus, Par, Bit	10,000,000–20,000,000	1–2 GB	108	5,340 ($D2 \bowtie D3$)
D4	Large	Uniform	10,000,000–20,000,000	1–2 GB	600	300 ($D4 \bowtie D4$)
D5	Large	Real	1,000,000–50,000,000	100 MB - 7 GB	94	431 ($D5 \bowtie D5$)

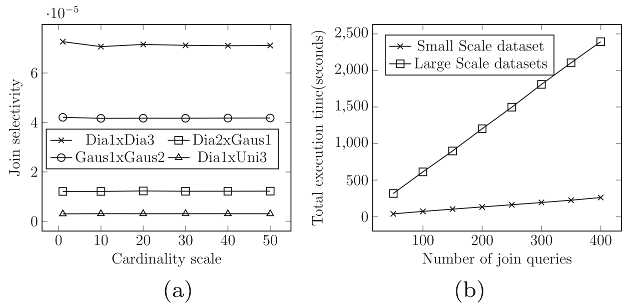

Fig. 5 Join selectivity and execution time in different cardinality scales

Table 5 Different possibilities of feature sets

Name	Features of D_*				Feat./metrics of (D_i, D_j)			
	P_{size}	P_{dens}	P_{part}	P_{dist}	P_{mbr}	P_{ch}	σ_{JN}	σ_{MT}
FS_s	✓	✓						
FS_{sh}	✓	✓		✓	✓			
FS_{shp}	✓	✓	✓	✓	✓	✓		
FS_{all}	✓	✓	✓	✓	✓	✓	✓	✓

MBR test selectivity, the best algorithm, or the total running time. In order to highlight the impact of the different features we define four possible training sets as shown in Table 5:

1. FS_s: the feature set that includes statistical-based features;
2. FS_{sh}: the feature set that includes FS_s and histogram-based features;
3. FS_{shp}: the feature set that includes FS_s, FS_{sh} and partitioning-based features;
4. FS_{all}: the feature set that include FS_{shp}, join cardinality, and number of MBR tests of all join algorithms.

The tabular data would be fed into an efficient regression or classification models, which will be discussed in Sect. 4.

6 Models

We propose a set of models for evaluating the metrics introduced in Sect. 5.1.3. Some of them are based on theoretical estimation formulas, others are the result of a training process. In each subsection we consider both previous approaches proposed in literature, used as baselines in experiments, and the models we propose in this paper. As highlighted in Fig. 3, M1 and M2 are used whenever users need to estimate the cost of computing the join in terms of result size (M1) and number of rectangle overlap tests (M2); this can be useful for generating an optimal query plan when the join is one of the operation to perform in a complex expression, for example, an analytical query. On the other hand, M3, M4 and M5 are used when users want to execute the join need to choose the partition strategy (M3), generating the set of join tasks to be executed on the cluster, and in which order each task has to process the set of geometries assigned to it (M5). M4 is used only for PBSM. In particular, M4 estimates the best number of partitions for PBSM algorithm, thus it already assumes that M3 chose PBSM as the best join strategy. In addition, M5 suggests the plane-sweep join strategy, regardless which algorithm was chosen previously by M3. Regarding the fact that we have tested M3 and M5 separately, we can observe that the improvement that M5 can produce in the join execution is orthogonal with respect to the alternative partition strategies, i.e. each strategy can be improved in the same way by M5 without changing the ranking generated by M3.

6.1 Join selectivity estimation models

Join selectivity (σ_{JN}) is defined as the ratio between the actual number of pairs produced by the join operation and the total number of pairs produced by the cross product. When join selectivity is estimated, the corresponding join cardinality can be obtained by multiplying for the size of the cross product.

Theoretical formula for σ_{JN}: as proposed in [4] and [9], given two datasets D_i and D_j with the corresponding features of group *Density* and *Overlapping*, the following estimates can be applied:

$$\sigma_{JN}(D_i, D_j) \approx \frac{1}{area_{\cap}}(mbrArea^{avg}(i)\,Area_j + mbrArea^{avg}(j)\,Area_i + (len_x^{avg}(i)len_y^{avg}(j) + len_x^{avg}(j)len_y^{avg}(i))\,Area_i Area_j)$$

where $area_{\cap}$ is the overlap area. In the provided source code, both the code and the results of its application to the chosen collection of datasets are available.

Along with the theoretical formula, we introduce machine learning based models to estimate the join selectivity model. In particular, we build a random forest regression model ($M1$) that takes the extracted features as the input and predict the join selectivity of a query. Our model can achieve up to 4.49% of the mean absolute percentage error (MAPE), which is pretty good when compared to the 35% error of the theoretical formula.

6.2 MBR tests selectivity models

Also for the MBR tests selectivity (σ_{MT}), which is the ratio between the actual number of MBR tests performed by the join operation and the cardinality of the cross product, we propose a theoretical baseline and a machine learning model.

Theoretical formula for σ_{MT}: as proposed in [9, 34], given two datasets D_i and D_j with the corresponding features of group *Size*, *Density*, *Partitioning & indexing* and *Overlapping*, the following estimate of MC (i.e. MBR tests count) for each algorithm A_i can be applied:

$$MC(D_i, D_j, A_i) \approx (n_i log(n_i + n_j) + n_j log(n_i + n_j)) \times \#combSplit_{A_i}$$

where $n_* = \frac{\#geo_*}{\#splits_*}$ and $\#combSplit_{A_i}$ is an estimate of the number of combined splits generated by algorithm A_i. For the algorithms that build a common grid, the value $\#combSplit_{A_i}$ is substituted by the number of non-empty cells of the common grid and n_* by the average number of geometries in the cells of the common grid. An estimate of σ_{MT} is derived by dividing the value of MC by the cardinality of the cross product.

Similar to the join selectivity estimation models, we proposed machine learning based models to estimate the MBR tests selectivity for distributed spatial join query. Since each join algorithm has a specific strategy, thus the number of MBR tests is an algorithm-dependent metric. Based on this fact, we build four separate models to predict the MBR tests selectivity for four spatial join algorithms (BNLJ, PBSM, DJ, REPJ). Each model is a random forest regressor that takes the extracted features as the input, then predicts the MBR tests selectivity. Our experiments show that the proposed models can achieve up to 1.77%, 1.25%, 4.5%, 3.3% MAPE value for BNLJ, PBSM, DJ, and REPJ, respectively. These performance is much better when compared to the theoretical formula above.

6.3 Algorithm selection models

We use three models that predict the best partitioning algorithm in terms of running time. The first model is the rule-based query optimizer for spatial join proposed in paper [52]. In short, the rule-based algorithm selects the most suitable partitioning algorithm based on heuristics and several rules. We implement this baseline algorithm as a Python function and share this function in our Github repository [59].

The function above works with an assumption that the two input datasets were spatially partitioned. The drawback of this baseline algorithm is that it is not very accurate and it never considers BNLJ as a potential join algorithm to execute.

The second model is given by the theoretical cost model proposed in [9]. Regarding this approach we can provide only some precalculated predictions regarding the chosen collection of synthetic and real datasets (see the Github repository [59]). Notice that, this model requires some additional features regarding the cluster used for the join algorithm executions, such as the number of nodes, the maximum number of mappers and reducers that can be instantiated and the relative measure of the cost of a CPU comparison operations w.r.t. the cost for local and network read/write operations. Thus, a direct comparison of this model, using other datasets, with our proposed solution is not possible.

The third model, that we propose in this paper, is a random forest classification model that takes the extracted features as the input, then predicts the best spatial join algorithm in terms of running time. We train and test the algorithm selection models using the dataset described in Sect. 5.2. The experiments section shows that the proposed machine learning model outperform other rule-based and theoretical models in the partitioning algorithm selection problem.

6.4 Tuning model: PBSM multiplier

The split factor for PBSM join algorithm defines the dimension of the grid that is used to partition the union of both input datasets. A baseline estimation approach can be based on the size of the two datasets $size(D_1)$ and $size(D_2)$ and the split size B:

$$dim_u = \left\lceil \sqrt{\frac{size(D_1) + size(D_2)}{B}} \right\rceil$$

In this case the suggested grid for partitioning the union of the input datasets will be of dimension: $dim_u \times dim_u$. This solution is only effective in the case that the considered input datasets are uniformly distributed. If they are skewed, this

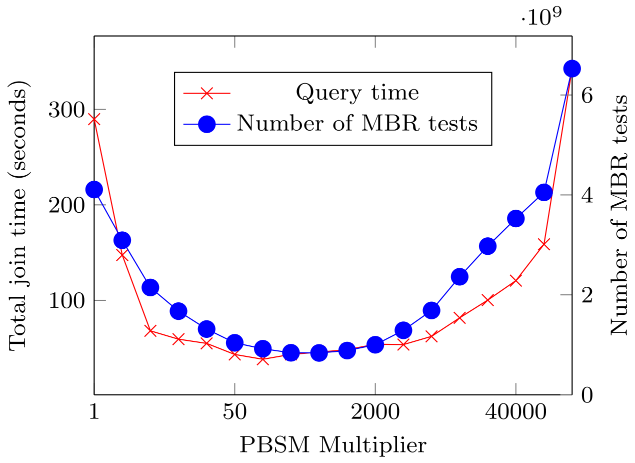

Fig. 6 Correlation between PBSM multiplier, total running time and number of MBR tests

splitting scheme produces unbalanced units of work. In order to produce a more balanced workload for distributed nodes, we increase the number of grid cells (and therefore of units of work or tasks) by introducing a new parameter, namely *PBSM Multiplier* α, which is an integer number. Introducing α the split factor will be computed as follows:

$$dim = \left\lceil \alpha \sqrt{\frac{size(D_1) + size(D_2)}{B}} \right\rceil$$

Figure 6 shows the total running time and number of MBR tests for spatial join queries in skewed datasets, with different values of PBSM Multiplier. We can observe two extreme cases of how to choose the PBSM Multiplier. If $\alpha = 1$, the number of tasks is the minimum number that guarantees each task will process an amount of B data size. Intuitively, this would not be the good strategy when data is skewed, then the advantages of distributed processing is not utilized. In the other place, if α is very large, i.e. 100, 000, the number of tasks is huge, which also creates the overhead of network communication and aggregation. In summary, we should not choose a very small or very large value for the PBSM Multiplier parameter.

Figure 6 also shows that there is a possibility to find the best PBSM Multiplier value for a join operation. In particular, there always exists a value of PBSM Multiplier so that the query running time or number of MBR tests is minimum. The main challenge here is this optimal value is different for each specific join inputs. In other words, the optimal PBSM Multiplier value depends on join input datasets. Motivated by this observation, we aim to build a machine learning model that is able to predict the PBSM Multiplier α according to the characteristics of the input datasets. In particular, we built a regression model M_4 that takes the features in Table 5 as the input, then predicts the best value of PBSM Multiplier α as the output. The detailed experimental results will be discussed in Sect. 7.6.1.

Table 6 Running time of spatial join when the plane-sweep algorithm sort by x-axis and y-axis

	PlaneSweep x-axis	PlaneSweep y-axis
Case 1	23.840000	31.550000
Case 2	29.376537	79.117921
Case 3	82.018803	27.477922
Case 4	80.052921	20.988435

6.5 Tuning model: plane sweep axis in local join operations

In general, most of distributed spatial join algorithms work in two main phases. Phase 1 can be called *global join*, that produces a list of pairs of partitions which are overlapping and potentially contain join results. Phase 2 can be called *local join*, which runs on each pair of overlapping partitions to produce a list of pairs of objects that satisfy the join predicate, commonly intersecting objects. The local join algorithm is mostly implemented with the same technique regardless of what the spatial join algorithm is, PBSM, BNLJ, RepJ or DJ. One common local join algorithm is plane-sweeping algorithm, which is first introduced at [50]. In brief, the plane-sweeping algorithm works in a sort-merge mechanism. First, it sorts the spatial objects of two input partitions based on one axis x or y. Second, it scans over two sorted list of objects to produce overlapping pairs with the merge-sort like scanning process. At the end of the process, we would get a list of intersecting objects with the average running time $O(n\,log(n))$, with n representing the total number of object of two input partitions.

As plane-sweeping algorithm is the basic operation of many distributed spatial join algorithms, it would be very beneficial if we can improve its performance. As we mentioned, there are two way to sort objects in plane-sweeping algorithms: sort by x-axis or sort by y-axis. In practice, people choose default axis without the awareness of how the input datasets look like. In fact, the sorting criteria plays an important role in plane-sweeping algorithm performance. Table 6 shows the running time of spatial join when we sort the objects by x-axis and y-axis in four different synthetic pairs of join inputs. We can observe that the choice of sort by x-axis or y-axis can make the running time up-to three times faster than the other way. Motivated by this observation, we aim to build a prediction model M_5, which takes the features from the input partitions, then suggests the sorting criterion that promises the better local join running time. In other words, M_5 will suggest the best axis for each pair of partitions to join. As each local join operation is optimized, we expect that the final running time of the spatial join operation would be minimized as well. The detailed experimental results will be discussed in Sect. 7.6.2.

7 Experiments

This section provides an experimental evaluation to measure the feasibility and accuracy of the proposed models. The experiments are designed to answer the following research questions:

1. Can machine learning models outperform hand-crafted theoretical models?
2. Do the proposed features catch all aspects of spatial join?
3. Is the proposed model generic enough to be applied on a dataset that was not in the training set?
4. How accurate is the model in choosing a spatial join algorithm to run?

7.1 Experimental setup

We implement four spatial join algorithms with their original design on Beast [16] - a Spark based system for big spatial data management. Beast is deployed on a Spark 3.0 cluster of one master node with 128GB RAM and 2×8-core Intel Xeon CPU E5-2609 v4 @1.7GHz, and 12 executor nodes each with 2×6-core Intel Xeon E5-2603 @1.7GHz and 10TB HDD.

The synthetic data is generated using the open-source Spider generator [35, 65]. The real data is downloaded from UCR-Star [30]. We chose the OSM buildings [22], lakes [21], and roads [23] datasets, and US Census linearwater [12], edges [11], and faces [13] datasets. Unless otherwise mentioned, models are trained only on synthetic data while the test data contains a mix of synthetic and real data. The detailed information of experimental datasets are described in our technical report [62].

As a baseline, we use the theoretical models proposed in [9, 34, 52]. To measure the accuracy of join selectivity model M1 and MBR test selectivity model M2, we use mean absolute error (MAE) and mean absolute percentage error (MAPE) which are calculated as following.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i|$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - x_i|}{x_i}$$

where y_i is the estimated value and x_i the true value.

For the algorithm selection model, we use the accuracy metric to evaluate how good the classifier can choose the best algorithm. To account for the cases where multiple algorithms provide almost the same running time, we also measure the MAPE between the running time of the selected algorithm and the best algorithm. A lower value of MAPE indicates a better selection model. Notice that we test each model separately since they address different problems, e.g., join selectivity and selection of partition algorithm, so an end-to-end comparison is not suitable.

The implementation of both the baselines and proposed models are publicly available at our Github repository [59]. In particular, we provide the options to use either Decision Tree or Random Forest algorithms to train the proposed classification and regression models. We use these classical algorithms because they are proven to be more appropriate with tabular data than deep learning based models. The maximum number of features is 26 as described in Table 3. The maximum tree depth is 8 in the current implementation [59]. The maximum number of training data points for proposed models is 1024. As the results, the training time is relatively small, mostly 10–20 s, and is an one time cost. Hence, we decided to not report the training time of proposed model in this paper so that we can focus on other important metrics.

7.2 Baseline methods

In order to compare the accuracy of the proposed model with previous work and show the improvements of the proposed approach, we define the following baseline methods.

- *Baseline B1 for the join selectivity model M1:* we consider the well-known spatial join selectivity estimation formula first proposed in [4] and also adopted in the theoretical model for spatial join algorithm ranking proposed in [9, 52]. This formula is defined for uniformly distributed datasets. To verify the accuracy of B1 for uniformly distributed data, we tested it on various datasets as reported in Table 7. As shown in the table, B1 provides good accuracy when applied on uniformly distributed dataset (column $MAPE_{JS}$), but the performance deteriorates when applied on skewed and non-completely overlapping datasets.
- *Baseline B2 for the MBR test selectivity model M2:* we consider the estimation of the theoretical model for ranking spatial join algorithms proposed in [9]. Notice that the theoretical model has a different goal, i.e. to predict the ranking of the spatial algorithms and not the number of MBR tests performed by them. The number of MBR test was used in [9] as an intermediate estimate to evaluate the relative position of the algorithms in the ranking, thus it is very imprecise when compared to the real values computed in the experiments. However, to the best of our knowledge there is no other approach in literature that try to estimate this parameter.

Table 7 Accuracy of the theoretical models (B1, $B3_2$) for join selectivity and the best two algorithms in ranking

Distribution	MBR overlap	MAPE_{JS}(B1)	Acc($B3_2$)
Uniform	≥ 90	1.7%	72%
Skewed	$\geq 90\%$	25%	64%
Skewed	$< 90\%$	$\gg 200\%$	57%

Table 8 Effect of feature selection to join selectivity estimation models

Feature Set	Model	MAPE (%)	MAE
% FS_s	B1	35	2.87×10^{-5}
FS_s	M1	23	1.06×10^{-5}
FS_{sh}	M1	4.49	2.36×10^{-6}

- *Baseline for algorithm selection model M3*: we compare our work with the rule-based model proposed in [52] (called $B3_1$) and the cost-based model proposed in [9] (called $B3_2$). The theoretical models were designed for uniformly distributed datasets, thus, as shown in Table 7 (column $Acc(B3_2)$, the accuracy of $B3_2$ in predicting at the best or the second best join algorithm decreases quickly considering non uniform datasets.

7.3 Feature selection

This experiment shows how the proposed feature selection affects the accuracy of the proposed models. In general, we expect that the more complex and intricate features will produce more accurate models. In each experiment, we compare three variations of proposed models M1, M2 and M3 with the three feature sets FS_s, FS_{sh}, and FS_{shp}. The tabular data is collected from 7140 join queries on synthetic datasets with different data distributions.

Table 8 shows the evaluation of join selectivity model (M1) and the baseline (B1) when using features sets FS_s and FS_{sh}. We do not use FS_{shp} since the partitioning information is not relevant to the join selectivity. First, we evaluate the performance of baseline method B1 from a previous work [4] on the feature set FS_s. Since B1 is designed to work with the join of uniform dataset, its MAPE value is pretty high. In addition, the equations of B1 use only the features available in the feature set FS_s. Given the same feature set FS_s, a random forest regression model M1 is significantly better than the baseline B1 gaining 12% in terms of MAPE. Finally, if we feed FS_{sh} to M1, the MAPE value (4.49%) is significantly reduced when compared to other models. This low MAPE value indicates that a regression random forest model with informative features can efficiently predict the join selectivity of a join query and outperform hand-crafted models.

Table 9 shows the efficiency of MBR selectivity (M2) and the baseline (B2) when using features sets FS_s, FS_{sh}, and FS_{shp}. Since we have a separate model for each of the four algorithms, we measure the performance of each of them separately. Overall, the baseline model B2 cannot work well with this problem, and its MAPE and MAE values are almost unacceptable. Keep in mind that B2 was not designed to accurately measure the MBR selectivity but it is roughly calculated at an intermediate step. In contrast, the proposed random forest regression model M2 can achieve very good values in terms of both MAPE and MAE values. We can also observe the downtrend of these metrics when there are more meaningful features fed into the model. We do not see much improvement for the models of BNLJ and PBSM, but noticeable improvement for the models of DJ and REPJ, which use the partitioning information in their join strategies. This observation further confirms our conjecture that machine learning models can outperform theoretical models when given enough features that describe the input datasets.

Table 10 shows the accuracy of algorithm selection (M3) and the baselines ($B3_1$ and $B3_2$) when using different features sets. We use two metrics to evaluate the efficiency of these algorithm selection models. The accuracy measures the percentage at which the model correctly estimates the best algorithm. The MAPE value measures the percentage of the difference between the running time of predicted algorithm and the actual best algorithm. The results show that $B3_1$ and $B3_2$ are not able to predict the best algorithm well, when their accuracy is too low, and the MAPE value is unreasonable. They are slightly better than a complete random choice which would yield 25% accuracy. The reason for this poor performance is that these models were mainly designed for uniformly distributed data and do not account for the complex spatial distributions. On the other hand, the proposed random forest classifier M3 can provide up to 82% accuracy and 7.4% MAPE value. The highest accuracy is produced with the model that uses all the proposed features. In addition, Fig. 7 show the confusion matrices of the baseline $B3_2$ and the proposed M3, respectively. This figure shows that the baseline get a decent accuracy (around 60%) for the PBSM and REPJ algorithms but get really confused about the other two algorithms, BNLJ and DJ.

In order to demonstrate the capability of our proposed framework on large real-world datasets, we carry out an additional experiment as follows. We wanted to answer the question “find all the intersecting roads in the world”. We can run a self-join operation on OpenStreetMap’s roads dataset, namely OSM2015/roads [23] to answer this question. OSM2015/roads contains 72 millions records with size 24.3 GB. First, we collect the dataset’s features and feed them

Table 9 Effect of feature selection on MBR selectivity estimation models B2 and M2

Feature Set	Model	BNLJ MAPE (%)	BNLJ MAE	PBSM MAPE (%)	PBSM MAE	DJ MAPE (%)	DJ MAE	REPJ MAPE (%)	REPJ MAE
FS_s	B2	518	1.13×10^{-2}	1332	7.18×10^{-4}	386	4.36×10^{-4}	183	7.82×10^{-4}
FS_s	**M2**	15	6.64×10^{-4}	**1.25**	$\mathbf{3.99\times10^{-5}}$	6.8	11.09×10^{-5}	7.5	8.53×10^{-5}
FS_{sh}	**M2**	1.8	$\mathbf{6.13\times10^{-5}}$	1.42	4.48×10^{-5}	5.4	8.25×10^{-5}	3.9	4.17×10^{-5}
FS_{shp}	**M2**	**1.77**	6.22×10^{-5}	1.39	4.38×10^{-5}	**4.5**	$\mathbf{6.95\times10^{-5}}$	**3.3**	$\mathbf{3.45\times10^{-5}}$

The best results are in bold

Table 10 Effect of feature sets on algorithm selection models

Feature Set	Model	Accuracy (%)	MAPE (%)
FS_{shp}	$B3_1$ [52]	33.7	122.8
FS_s	$B3_2$ [9]	32.2	80.5
FS_s	M3	71	13.9
FS_{sh}	M3	79	7.9
FS_{shp}	M3	81	7.5
FS_{all}	M3	82	7.4

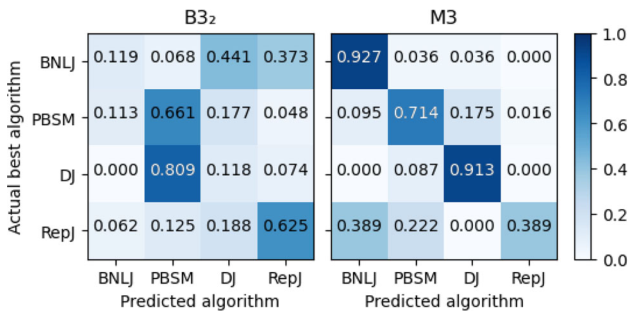

Fig. 7 The confusion matrix of the algorithm selection model

Table 11 Self-join running time (in seconds) of different spatial join algorithms on OSM2015/roads dataset

PBSM	DJ	RepJ	BNLJ
1258.63	745.31	**717.51**	740.95

The best results are in bold

to the Algorithm Selection Model (described in Sect. 6.3). The model suggests that RepJ is the best join algorithm in terms of running time. After that, we ran all spatial join algorithms with the given input. The running time for each algorithm are shown in Table 11. This result verified that RepJ is actually the fastest join algorithm. This experiment showed that Algorithm Selection Model is able to suggest database engine to choose the optimized join algorithm based on given inputs.

On the contrary, the proposed model is generally accurate, especially for BNLJ, PBSM, and DJ, but performs less accurately for the REPJ. We plan to address this issue in the future

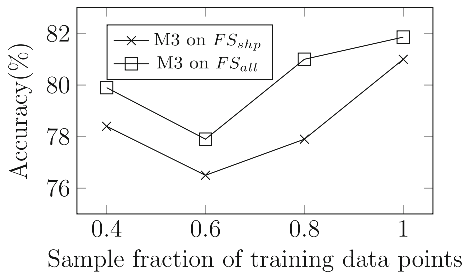

Fig. 8 Performance of M3 for the FS_{shp} and FS_{all} feature sets as the training set size increases

but the results are very encouraging to include the proposed optimizer into existing spatial database systems.

Finally, we wanted to take a closer look into the difference between the feature set FS_{shp} and FS_{all}. In this experiment, we fix the test set and gradually increase the training set from 40% to 100% of all the available training data. Figure 8 shows that the model performs generally better with the feature set FS_{all} even when the training set is small. This indicates that the join selectivity and MBR selectivity, which are added in FS_{all}, help in improving the model performance.

7.4 Training set generation

This experiment shows the effect of the training set on the accuracy of the models. Because it is not practical to construct training datasets that cover all of distributions, we have been trying to build a training set which can reflect the real-world datasets as much as possible. Our goal is to show that the more distributions we have in the training set, the more accurate the model becomes for a dataset with any distribution. To verify that, we gradually increase the number of synthetic distributions in the training set, from 1 to 5, and measure MAE of join selectivity model M1 and MBR tests selectivity model M2. For fairness, when we limit the number of distributions, we try all combinations and take the average. For example, for two distributions, we try $\binom{5}{2} = 10$ combinations and report their average. Finally, we add the data with real distribution and consider the total number of distributions being six. These six distributions well represent most available spatial datasets as shown in [35, 65]. Notice that this is

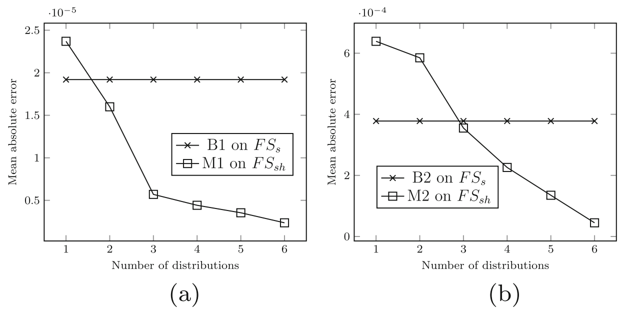

Fig. 9 Effect of training distributions in M1 and M2 model

the only experiment that uses real data for model training to test how far it can enrich the model. Figure 9 shows the efficiency of the baseline models B1 and B2 and the proposed models M1 and M2 when the number of distributions varies, while the test dataset is fixed. Since there is no training process for B1 and B2, their MAE values are fixed. In contrast, the MAE values for M1 and M2 improves as the number of distributions increases. This behavior indicates that adding more distribution to the training data helps in improving the model's performance. Moreover, there is a slight difference of MAE value between the training data with all five distributions and the training data with real dataset's join results. This small gap verifies that the proposed model can generalize to datasets with distributions that were not included in the training set. In other words, the proposed model can provide some level of out-of-distribution generalization for real datasets whose distributions are not exactly those provided by synthetic datasets. Furthermore, users can still add more datasets with diversified distributions to enrich the model's training data for their own models.

7.5 Spatial join cost estimation model

This experiment measures the performance of the proposed linear regression model M4 which estimates the algorithm running time. Input features are only the result size and number of MBR tests, obtained by multiplying σ_{JN} and σ_{MT} (estimated by M1 and M2) by the cardinality of the cross product. Our goal is to show that these two features accurately estimates the overall running time. Additionally, since this is a linear model, it can be trained on a very small dataset. This allows system designers to use pre-trained M1 and M2 models together with a very small training set on their hardware to measure the spatial join query performance. Table 12 shows that the regression score of these models is very good for the four algorithms. In addition, we plot the estimated running time with predicted running time for the different linear regression models in Fig. 10. In this figure, we show the models with highest score (BNLJ) and lowest score (PBSM).

Table 12 Regression score of the linear model between join result, number of MBR tests and join execution time

Algorithm	Regression Score
BNLJ	0.86
PBSM	0.78
DJ	0.82
REPJ	0.80

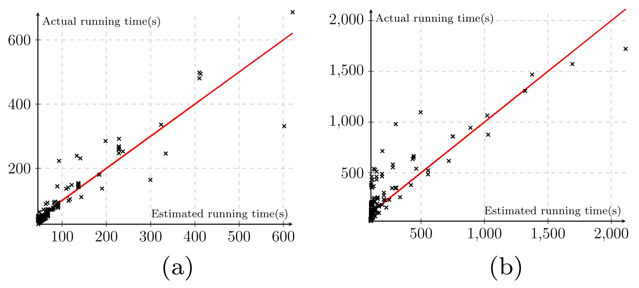

Fig. 10 Spatial join running time estimation with M4

7.6 Tuning models

In this section we evaluate the capability to further improve the performances of the spatial join by tuning two parameters: the PBSM multiplier and the choice between sorting of geometries on the x or y axes during the execution of the plane sweep algorithm. The following two subsections illustrate the results obtained by training two models for tuning them.

7.6.1 PBSM multiplier

Training data generation. In Sect. 5.2, we validated that the running time of a join operation is highly correlated with the number of MBR tests for that operation. In addition, the join operation between large datasets is very expensive. Therefore, we decided to generate the training data using medium scale datasets. In particular, we join the datasets with size from 10 MB to 30 MB, and use the number of MBR tests as a proxy to the running time. The join operation with minimum number of MBR tests will likely be the join with least running time.

Given a pair of datasets, we use a simple binary-search-like algorithm to find the optimal PBSM multiplier. The algorithm works as follows:

1. Start with PBSM Multiplier with value 1. Execute the join and record the number of MBR tests.
2. Increase PBSM Multiplier by 10 times. Execute the join and record the number of MBR tests. Repeat this step and stop when the number of MBR tests start increasing.

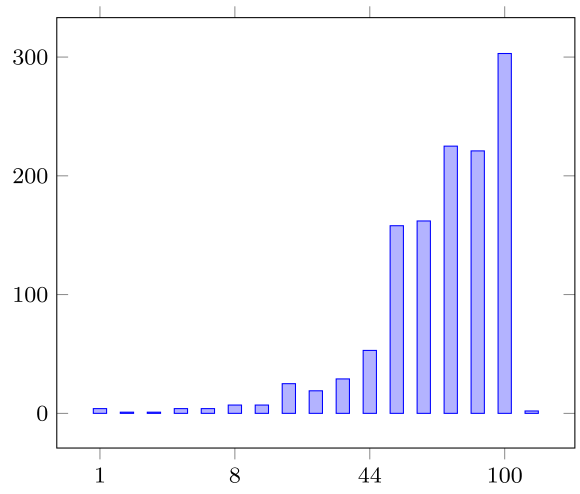

Fig. 11 Frequency of the best PBSM Multiplier value

Table 13 PBSM-Multiplier model comparison

	MAE	MAPE	MSE
Baseline	23.88	1.43	1054.56
Random Forest	10.03	0.35	202.25

3. Choose the PBSM Multiplier which is the middle of the last two value. Execute the join and record the number of MBR tests.
4. Stop the process after 3 loops. Record the PBSM Multiplier that give the minimum number of MBR tests.

Model training and testing. The PBSM multiplier prediction is a regression problem. In which we use the features in Sect. 5.2 as the model features, and the integer number of PBSM multiplier as the label. We simply use a random forest model to train the PBSM multiplier prediction model. The baseline value is chosen equal to 100 as this is the most common best value of PBSM Multiplier as shown in Fig. 11. The test results is shown in Table 13. The experimental results show that the learning-based model provides a significant better estimation for the best PBSM Multiplier value, which promises a shorter running time for spatial join operations.

7.6.2 Plane sweep axis in local join operations

Training data generation. In this experiment, we are going to run a spatial join operation on *OSM Roads* and *TIGER Linear Water* datasets. First, we utilize the PBSM spatial join algorithm to split the join inputs into partitions, then we produce the list of overlapping partitions, which potentially contain the join results. After that, we run both plane-sweeping algorithms that sort the objects by x-axis and y-axis in all the overlapping pair. We record the running time of each operation to evaluate whether the sorted by x-axis or y-axis is faster. At the end of this process, we collect the label of training data points, which can be either x-axis is better or y-axis is better. To build the features for each data point, we reused the listed feature FS_{sh} in Table 5. In addition, we also collect other features of two input partitions such as intersection percentage and Jaccard similarity of two partitions by x-axis or y-axis. Note that we do not reuse the feature values calculated for the entire dataset but we recalculate them for the data within the current partition since each partition could possibly make a different decision about sorting by x or y. This requires a single scan over the data in this partition which is a negligible cost as compared to sorting.

Table 14 PBSM-Multiplier model comparison

	Accuracy
Baseline	0.5
Random Forest 1	0.7297
Random Forest 2	0.7972

Model training and testing. The plane-sweep axis selection problem is a classification problem. We decide to build two Random Forest classification models based on the generated training data. The first model is built based on FS_{sh} features in Table 5. The second model is built with FS_{sh} features and the features of intersection percentage and Jaccard similarity of two partitions by x-axis or y-axis. We choose the baseline as a random choice that chooses either x-axis or y-axis as sorting criteria. Obviously, the accuracy of the baseline is 0.5. The experimental results in Table 14 shows that the second Random Forest model built with additional features gives us the best accuracy. This is expected since the relationship by x-axis and y-axis is significantly important for the performance of local join operation.

8 Conclusion

This paper presented a framework for supporting spatial join processing by providing different models based on machine learning for cost estimation. It is able to choose the best distributed spatial join algorithm given two input datasets. It breaks down the cost estimation into several modules.

The first module estimates the cardinality of the spatial join result which is an algorithm independent metric. Second, it estimates the number of MBR tests done by each algorithm which is a machine-independent but algorithm-specific metric of performance. The third module estimates the best algorithm. Then, the fourth model estimates the number of partitions, if needed. The final model chooses the best sorting direction for the plane-sweep local join algorithm.

To train these models, we used a synthetic data generator that produces thousands of datasets with various distributions to train the model on the different aspects of spatial data. We further enrich the training data by some real datasets to improve the diversity. We showed that training on synthetic and small/medium size datasets can speed up the model setting and still produce accurate results, also when the test is performed on large datasets.

Our experimental evaluation shows the effectiveness of the proposed method in estimating: (i) the join cardinality, which is an algorithm- and machine-independent metrics for measuring the cost of spatial join operation; (ii) the number of MBR tests performed by the join operation, which is a machine-independent metrics; (iii) the best algorithm to be applied for executing the join in a distributed environment; (iv) some tuning parameters that allows to obtain be best running time in the join execution both at global level (partitioning) and local level (plane-sweep algorithm application at each node of the cluster).

In the future, we plan to further extend this model by taking into account the hardware specification to estimate a more accurate result. We also want to reconsider the feature extraction phase in order to improve the accuracy of the models. In addition, refinement phase optimization could be a good opportunity to improve join performance in case of datasets with complex geometries. Another interesting work to consider for the future is to train correlated models together to capture their intertwined relationship or spatial join with 1D partitioning [5, 56].

Author Contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by all authors. The first draft of the manuscript was written by all authors and we commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding This work is supported in part by the National Science Foundation (NSF) under Grants IIS-1954644, IIS-1838222 and CNS-1924694 and by "Progetto di Eccellenza" of the Computer Science Dept., Univ. of Verona, Italy. Tin Vu performed this work while at UC Riverside.

Declarations

Conflict of interest Editor-in-chief or editorial board member of the VLDB Journal with whom they have a conflict-of-interest along with the nature of the conflict: Yuanyuan Tian Microsoft Gray Systems Lab, USA. All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

1. Acharya, S., Poosala, V., Ramaswamy, S.: Selectivity estimation in spatial databases. In: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, pp. 13–24 (1999)
2. Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., Saltz, J.: Hadoop-gis: A high performance spatial data warehousing system over mapreduce. In: Proceedings of the VLDB Endowment International Conference on Very Large Data Bases, vol. 6. NIH Public Access (2013)
3. An, N., Yang, Z., Sivasubramaniam, A.: Selectivity estimation for spatial joins. In: ICDE, pp. 368–375 (2001)
4. Aref, W., Samet, H.: A cost model for query optimization using R-Trees. In: GIS, pp. 60–67 (1994)
5. Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., Vitter, J.S.: Scalable sweeping-based spatial join. In: VLDB, vol. 98, pp. 570–581. Citeseer (1998)
6. Baig, F., Vo, H., Kurc, T., Saltz, J., Wang, F.: Sparkgis: Resource aware efficient in-memory spatial query processing. In: Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 1–10 (2017)
7. Belussi, A., Faloutsos, C.: Self-spacial join selectivity estimation using fractal concepts. ACM TIS **16**(2), 161–201 (1998)
8. Belussi, A., Migliorini, S., Eldawy, A.: Detecting skewness of big spatial data in spatialhadoop. In: Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '18, pp. 432-435 (2018). https://doi.org/10.1145/3274895.3274923
9. Belussi, A., Migliorini, S., Eldawy, A.: Cost estimation of spatial join in spatialhadoop. GeoInformatica **24**, 1021–1059 (2020). https://doi.org/10.1007/s10707-020-00414-x
10. Belussi, A., Migliorini, S., Eldawy, A.: Skewness-based partitioning in SpatialHadoop. ISPRS IJGI **9**(4), 201:1-201:19 (2020)
11. Bureau, U.C.: All tiger lines (2019). https://doi.org/10.6086/N1P55KJS
12. Bureau, U.C.: Linear hydrography (2019). https://doi.org/10.6086/N1QF8QW4
13. Bureau, U.C.: Topological faces (polygons with all geocodes) (2019). https://doi.org/10.6086/N19021TG
14. den Bercken, J.V., Seeger, B., Widmayer, P.: The bulk index join: A generic approach to processing non-equijoins. In: M. Kitsuregawa, M.P. Papazoglou, C. Pu (eds.) Proceedings of the 15th International Conference on Data Engineering, Sydney, Australia, March 23-26, 1999, 257. IEEE Computer Society (1999). https://doi.org/10.1109/ICDE.1999.754937
15. Du, Z., Zhao, X., Ye, X., Zhou, J., Zhang, F., Liu, R.: An effective high-performance multiway spatial join algorithm with spark. ISPRS Int. J. Geo Inf. **6**(4), 96 (2017)
16. Eldawy, A., Hristidis, V., Ghosh, S., Saeedan, M., Sevim, A., Siddique, A., Singla, S., Sivaram, G., Vu, T., Zhang, Y.: Beast: Scalable Exploratory Analytics on Spatio-temporal Data. In: CIKM. ACM (2021)
17. Eldawy, A., Mokbel, M.F.: Spatialhadoop: A mapreduce framework for spatial data. In: 2015 IEEE 31st International Conference on Data Engineering, pp. 1352–1363. IEEE (2015)
18. Eldawy, A., Mokbel, M.F.: SpatialHadoop: A MapReduce framework for spatial data. In: ICDE, pp. 1352–1363 (2015)

19. Eldawy, A., Mokbel, M.F.: The era of big spatial data: a survey. Found. Trends Databases **6**(3–4), 163–273 (2016). https://doi.org/10.1561/1900000054
20. Eldawy, A., Mokbel, M.F.: Spatial join with hadoop. In: Shekhar, S., Xiong, H., Zhou, X. (eds.) Encyclopedia of GIS, pp. 2032–2036. Springer (2017). https://doi.org/10.1007/978-3-319-17885-1_1570
21. Eldawy, A., Mokbel, M.F.: All water areas in the world from openstreetmap (2019). https://doi.org/10.6086/N1668B70
22. Eldawy, A., Mokbel, M.F.: The boundaries of all buildings in the world as extracted from openstreetmap (2019). https://doi.org/10.6086/N1JW8BWH
23. Eldawy, A., Mokbel, M.F.: Roads and streets around the world each represented as a polyline extracted from openstreetmap (2019). https://doi.org/10.6086/N1XK8CK6
24. Eldawy, A., Mokbel, M.F., Al-Harthi, S., Alzaidy, A., Tarek, K., Ghani, S.: SHAHED: A mapreduce-based system for querying and visualizing spatio-temporal satellite data. 1585–1596. Seoul, South Korea (2015)
25. Estan, C., Naughton, J.F.: End-biased samples for join cardinality estimation. In: Proceedings of the 22nd International Conference on Data Engineering, ICDE, 20. IEEE Computer Society (2006). https://doi.org/10.1109/ICDE.2006.61
26. Estan, C., Naughton, J.F.: End-biased samples for join cardinality estimation. In: 22nd International Conference on Data Engineering (ICDE'06), pp. 20–20. IEEE (2006)
27. Faloutsos, C., Seeger, B., Traina, A., Traina, C.: Spatial join selectivity using power laws. In: SIGMOD, SIGMOD'00, 177-188 (2000)
28. Fornari, M.R., Comba, J.L.D., Iochpe, C.: Query optimizer for spatial join operations. In: GIS, pp. 219–226. ACM (2006)
29. Georgiadis, T., Mamoulis, N.: Raster intervals: an approximation technique for polygon intersection joins. Proc. ACM Manag. Data **1**(1), 1–18 (2023)
30. Ghosh, S., Vu, T., Eskandari, M.A., Eldawy, A.: UCR-STAR: tUCR spatio-temporal active repository. SIGSPATIAL Spec. **11**(2), 34–40 (2019)
31. Goodchild, M.F.: Citizens as voluntary sensors: spatial data infrastructure in the world of web 2.0. IJSDIR **2**, 24–32 (2007)
32. Gupta, H., Chawda, B.: ε-controlled-replicate: An improved controlled-replicate algorithm for multi-way spatial join processing on map-reduce. In: International Conference on Web Information Systems Engineering. Springer (2014)
33. Henke, N., et al.: The Age of Analytics: Competing in a Data-driven World. Tech. rep, McKinsey Global Institute (2016)
34. Jacox, E.H., Samet, H.: Spatial join techniques. ACM Trans. Database Syst. (TODS) **32**(1), 7 (2007)
35. Katiyar, P., Vu, T., Migliorini, S., Belussi, A., Eldawy, A.: SpiderWeb: A Spatial Data Generator on the Web. In: SIGSPATIAL. ACM (2020)
36. Kim, J., Hong, B.: Parallel spatial joins using grid files. In: Seventh International Conference on Parallel and Distributed Systems, ICPADS 2000, Iwate, Japan, July 4-7, 2000, 531–536. IEEE Computer Society (2000). https://doi.org/10.1109/ICPADS.2000.857739
37. Kipf, A., Kipf, T., Radke, B., Leis, V., Boncz, P.A., Kemper, A.: Learned cardinalities: Estimating correlated joins with deep learning. In: CIDR (2019)
38. Krishnan, S., Yang, Z., Goldberg, K., Hellerstein, J.M., Stoica, I.: Learning to optimize join queries with deep reinforcement learning. arXiv:1808.03196 (2018)
39. Leis, V., Radke, B., Gubichev, A., Kemper, A., Neumann, T.: Cardinality estimation done right: Index-based join sampling. In: Cidr (2017)
40. Leis, V., et al.: Cardinality estimation done right: Index-based join sampling. In: CIDR (2017)
41. Magdy, A., Alarabi, L., Al-Harthi, S., Musleh, M., Ghanem, T.M., Ghani, S., Mokbel, M.F.: Taghreed: a system for querying, analyzing, and visualizing geotagged microblogs, pp. 163–172. Dallas/Fort Worth, TX (2014)
42. Marcus, R., Negi, P., Mao, H., Tatbul, N., Alizadeh, M., Kraska, T.: Bao: Learning to steer query optimizers. In: SIGMOD (2021)
43. Marcus, R., Papaemmanouil, O.: Deep reinforcement learning for join order enumeration. In: R. Bordawekar, O. Shmueli (eds.) aiDM@SIGMOD, 3:1–3:4. ACM (2018)
44. Marcus, R.C., et al.: Neo: a learned query optimizer. PVLDB **12**(11), 1705–1718 (2019)
45. Ono, K., Lohman, G.M.: Measuring the complexity of join enumeration in query optimization. In: PVLDB, pp. 314–325 (1990)
46. Ortiz, J., Balazinska, M., Gehrke, J., Keerthi, S.S.: An empirical analysis of deep learning for cardinality estimation. arXiv preprint arXiv:1905.06425 (2019)
47. Patel, J.M., DeWitt, D.J.: Partition based spatial-merge join. In: Jagadish, H.V., Mumick, I.S. (eds.) Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996, pp. 259–270. ACM Press (1996). https://doi.org/10.1145/233269.233338
48. Patel, J.M., DeWitt, D.J.: Partition based spatial-merge join. SIGMOD Rec. **25**(2), 259–270 (1996). https://doi.org/10.1145/235968.233338
49. Pedregosa, F., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
50. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Springer (2012)
51. Ray, S., Simion, B., Brown, A.D., Johnson, R.: Skew-resistant parallel in-memory spatial join. In: Jensen, C.S., Lu, H., Pedersen, T.B., Thomsen, C., Torp, K. (eds.) Conference on Scientific and Statistical Database Management, SSDBM '14, Aalborg, Denmark, June 30 - July 02, 2014, 6:1–6:12. ACM (2014). https://doi.org/10.1145/2618243.2618262
52. Sabek, I., Mokbel, M.F.: On Spatial Joins in MapReduce. In: SIGSPATIAL (2017). https://doi.org/10.1145/3139958.3139967
53. Singla, S., Eldawy, A.: Flexible Computation of Multidimensional Histograms. In: SpatialGems. ACM (2020)
54. Sun, C., Bandi, N., Agrawal, D., El Abbadi, A.: Exploring spatial datasets with histograms. Distrib. Parallel Databases **20**(1), 57–88 (2006)
55. The Common Metadata Repository: The Foundation of NASA's Earth Observation Data (2017). https://earthdata.nasa.gov/the-common-metadata-repository
56. Tsitsigkos, D., Bouros, P., Mamoulis, N., Terrovitis, M.: Parallel in-memory evaluation of spatial joins. In: Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 516–519 (2019)
57. Twitter Usage Statistics. http://www.internetlivestats.com/twitter-statistics/ (2018). Visited on 15-Sep-2021
58. Vengerov, D., Menck, A.C., Zaït, M., Chakkappen, S.: Join size estimation subject to filter conditions. PVLDB **8**(12) (2015)
59. Vu, T.: A learning based framework for spatial join processing: estimation, optimization and tuning. https://github.com/tinvukhac/learned-spatial-join (2023)
60. Vu, T., Belussi, A., Migliorini, S., Eldawy, A.: Using deep learning for big spatial data partitioning. TSAS **7**(1), 3:1-3:37 (2020)
61. Vu, T., Belussi, A., Migliorini, S., Eldawy, A.: A Learned Query Optimizer for Spatial Join. In: ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL 2021. ACM (2021). https://doi.org/10.1145/3474717.3484217
62. Vu, T., Belussi, A., Migliorini, S., Eldawy, A.: Towards a learned cost model for distributed spatial join: Data, code & models. In: Proceedings of the 31st ACM International Conference on Information & Knowledge Management, pp. 4550–4554 (2022)

63. Vu, T., Eldawy, A.: R-grove: Growing a family of r-trees in the big-data forest. In: Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 532–535 (2018)
64. Vu, T., Eldawy, A.: R*-grove: balanced spatial partitioning for large-scale datasets. Front. Big Data **3**, 28 (2020)
65. Vu, T., Migliorini, S., Eldawy, A., Belussi, A.: Spatial data generators. In: 1st ACM SIGSPATIAL Int. Workshop on Spatial Gems (SpatialGems 2019), 7 (2019)
66. Xie, D., Li, F., Yao, B., Li, G., Zhou, L., Guo, M.: Simba: Efficient in-memory spatial analytics. In: SIGMOD, pp. 1071–1085 (2016)
67. Yang, Z., et al.: NeuroCard: one cardinality estimator for all tables. PVLDB **14**(1), 61–73 (2020)
68. Yu, J., Wu, J., Sarwat, M.: GeoSpark: a cluster computing framework for processing large-scale spatial data. In: SIGSPATIAL, pp. 70:1–70:4 (2015)
69. Yu, J., Wu, J., Sarwat, M.: A demonstration of geospark: A cluster computing framework for processing big spatial data. In: 32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016, pp. 1410–1413 (2016)
70. Zhang, S., Han, J., Liu, Z., Wang, K., Xu, Z.: SJMR: parallelizing spatial join with mapreduce on clusters. In: CLUSTER, 1–8. IEEE Computer Society, New Orleans, LA (2009). https://doi.org/10.1109/CLUSTR.2009.5289178