A Learned Query Optimizer for Spatial Join*

Tin Vu
Computer Science and Engineering
University of California, Riverside
Riverside, CA, USA
tin.vu@email.ucr.edu

Alberto Belussi
Sara Migliorini
Dept. of Computer Science
University of Verona, Italy
{alberto.belussi,sara.migliorini}@univr.it

Ahmed Eldawy
Computer Science and Engineering
University of California, Riverside
Riverside, CA, USA
eldawy@ucr.edu

ABSTRACT

The importance and complexity of spatial join resulted in many join algorithms, some of which run on big-data platforms such as Hadoop and Spark. This paper proposes the first machine-learning-based query optimizer for spatial join operation which can accommodate the skewness of the spatial datasets and the complexity of the different algorithms. The main challenge is how to develop portable cost models that take into account the important input characteristics such as data distribution, spatial partitioning, logic of spatial join algorithms, and the relationship between the two datasets. The proposed system defines a set of features that can all be computed efficiently for the data to catch the intricate aspects of spatial join. Then, it uses these features to train three machine learning models that capture several metrics to estimate the cost of four spatial join algorithms according to user requirements. The first model can estimate the cardinality of spatial join algorithm. The second model can predict the number of rough comparisons for a specific join algorithm. Finally, the third model is a classification model that can choose the best join algorithm to run. Experiments on large scale synthetic and real data show the efficiency of the proposed models over baseline methods.

CCS CONCEPTS

• **Information systems → Database management system engines**; • **Computing methodologies → Machine learning approaches**.

KEYWORDS

spatial join, query optimizer, machine learning, big data

ACM Reference Format:
Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2021. A Learned Query Optimizer for Spatial Join. In *29th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '21), November 2–5, 2021, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3474717.3484217

*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

SIGSPATIAL '21, November 2–5, 2021, Beijing, China

ACM ISBN 978-1-4503-8664-7/21/11.
https://doi.org/10.1145/3474717.3484217

1 INTRODUCTION

The rapid increase in the amount of big spatial data urged the research community to develop many systems to process this huge amount of spatial data, such as SpatialHadoop [11], GeoSpark [38], and many others [12]. One of the most important and challenging operations in all these systems is *spatial join*, which combines multiple datasets together based on their spatial features.

Spatial join is one of the most resource demanding operations in spatial databases and it becomes even more challenging with big data [5, 13, 31]. Since big spatial data systems run in a distributed environment, 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 encourages the researchers to develop many spatial join algorithms for big data [11, 21, 38]. However, this creates a complex query optimization problem to choose the best spatial join algorithm given the input datasets and hardware resources.

Traditionally, this query optimization problem has been addressed using theoretical cost models [1, 3, 18, 19]. With the rise of big-data, some of these cost models have been ported to MapReduce and similar systems [2, 31]. 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 query optimizers that rely on data-driven models for database operations including join [23, 24, 27, 28, 37]. 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 datasets are used for training and testing which limit the applicability of the produced models.

This paper proposes the first learned query optimizer for distributed spatial join on big data. This cost model can be abstracted as a complex function that estimates a single value, e.g., 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 that consists of three levels of application. 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* combines the predicted result size with other dataset attributes to build 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. Finally, the *third level* builds a classification model which is able to predict the best algorithm.

This proposed architecture gives system designers the flexibility of choosing the level that matches their application needs from the most general (level 1) to the most specific one (level 3). 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.

The proposed work is open source [1]. In particular, we implement the proposed models using *scikit-learn* [30] and train/test them on different datasets. To train the model, we use an open-source spatial data generator [36] 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 [20]. 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. Finally, we tested the end-to-end framework in predicting the best algorithm in terms of running time.

In summary, we make the following contributions:

(1) We design and implement the first machine learning based model which is able to predict the best algorithm for spatial join in terms of their running times.
(2) We break down the time prediction model into separate models that predict join selectivity, number of MBR tests, and fastest algorithm. These models are the building blocks for existing spatial join query optimizers.
(3) We carry extensive experiments to validate the advantages of the proposed models over the existing solutions.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 describes the process of our proposed cost model. Section 4 details the training and test process including data generation and preparation. Section 5 gives the results of our experiments. Finally, Section 6 concludes the papers and discusses future works.

2 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 [17, 25, 29, 33]. 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 [23, 37], join cost estimation [24, 26, 28], and join order enumeration [27]. These ML-based 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 is 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.

Join Type	Non-spatial	Spatial
Problems	Selectivity estimation, join cost, and join ordering	Selectivity estimation and join cost estimation
Theoretical	[17, 25, 29, 33]	[1–3, 18, 19, 31]
ML	[23, 24, 26–28, 37]	**This work**

Figure 1: Related work in join optimization

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 [2], 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 [3] and binary join [18]. 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 [31]. 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 [5] 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.

This paper proposes the first ML-based model for distributed spatial join for selectivity estimation and cost estimation. 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.

3 OVERVIEW OF SJML

This section gives an overview of the proposed Spatial Join Machine Learning (SJML) tool illustrated in Figure 2. 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. Thus, in the training phase, we use Spider [36], a standard

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

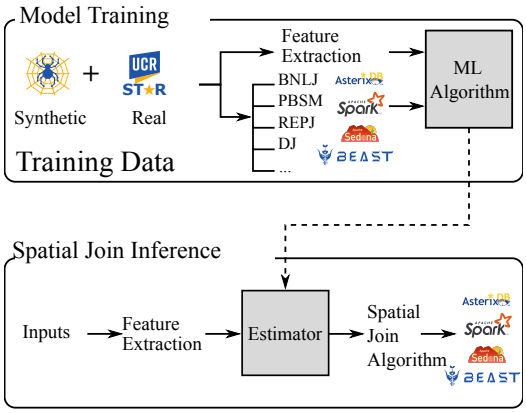


Figure 2: Overview of SJML

spatial data generator, to generate various datasets with diverse characteristics. For each pair of generated datasets, we extract a set of features that the machine-learning model can train on. We also run all available algorithms on each pair of datasets to measure their behavior with such pair of datasets. This 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.

This generic model can easily extend 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 model is built, we can use it to infer the best spatial join algorithm to run. Given a pair of datasets, we extract their features in the same way done in the training phase. We feed the features to SJML to choose the best algorithm to run.

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.

Model Training and Inference: We train three models that work in concert to solve the problem. The first model estimates the size of the spatial join result which is a very important algorithm-independent metric for the cost of spatial join. The second model estimates the total number of rectangle tests (i.e., intersection tests between the MBR of two geometries) in the algorithm which is an algorithm-specific but hardware-independent metric for the cost of spatial join. The third model combines these two metrics with other features to choose the best algorithm. The first two models are regression models while the third is a classification model.

Table 1: Summary of features

	Group	Feature	Description
Single dataset (D_*)	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}	$Area_i$	Percentage of overlap area occupied by dataset i or j
		$Area_j$	
		$JS_{i,j}$	Jaccard similarity between the two MBRs
	P_{ch}	e_0, e_2	Box counting for the convoluted histogram with base 0 and 2

4 MODEL TRAINING AND TESTING

The training process runs a supervised training step to build an ensemble of models that estimate the cost of spatial join. First, it runs a feature extraction step that takes two datasets and converts them into a fixed-size feature vector and 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 [36] to generate diverse synthetic datasets, and UCR-Star [20] to extract real datasets. Finally, we prepare the master dataset 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.

4.1 Feature Extraction

The spatial datasets that the spatial join operation processes are not directly usable by the machine learning models that we propose, since they expect a fixed-size feature vector not a variable size dataset. This step extracts a 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 one parallel scan over the data.

Table 1 summarizes all features we use in this paper and the details are given in this section. We divide the set of features into two groups, *single* and *combined* dataset statistics. Single dataset statistics are features that are extracted for each individual dataset separately. Combined statistics represent features that are computed

for the two datasets together. Finally, We collect a set of *metrics* from the execution of each spatial join algorithm that represent its cost.

4.1.1 Single Dataset Statistics. The following statistics are computed individually for each dataset D_* and they are not modified based on the join query, since they describe the dataset itself. We define them into four groups. The first two groups contain a set of associative and commutative functions that can be computed efficiently in one parallel scan. The third group is computed from a histogram over the data which can be computed exactly in one additional scan, or approximately in one scan that can be combined with the first two groups [32]. The fourth group is computed from the partition information and does not require scanning the actual records. Further details are provided in the following four definitions.

Definition 4.1 (Data size statistics - P_{Size}). Given a dataset D_*, we define:

- $\#geo(D_*)$: number of geometries (records) in D_*: $|D_*|$.
- $size(D_*)$: size of the dataset in bytes. $size(D_*) = \sum_{r \in D_*} size(r)$, where $size(r)$ is the size of a record in bytes.

Definition 4.2 (Data density statistics - P_{dens}). Given a dataset D_*, we define:

- $\texttt{MBR}(D_*)$: Minimum Bounding Rectangle of D_*. $\texttt{MBR}(D_*) = (\min x, \min y, \max x, \max y)$.
- $mbrArea^{avg}(D_*)$: given the MBR of geometries r in D_* ($mbr(r)$), it represents the average area of such MBRs: $mbrArea^{avg}(D_*) = \frac{\sum_{r \in D_*} area(mbr(r))}{|D_*|}$
- $len_x^{avg}(D_*)$ 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. $len_x^{avg}(D_*) = \frac{\sum_{r \in D_*} len_x(mbr(r))}{|D_*|}$. $len_y^{avg}(D_*)$ is calculated similarly.

After the above statistics are calculated, the histogram of the input is computed. We always create a high-resolution histogram of size 8192×8192. This histogram is used to calculate in particular the parameters E_0 and E_2 using the box-counting technique as described in [6]. The histogram is also used to calculate the lower-resolution histograms $H(D_*)$ that will be used for the computation of the convoluted histogram described shortly.

Definition 4.3 (Histogram-based statistics - P_{dist}). Given a dataset D_*, its high-resolution histogram h is generated and some metrics, first proposed in [4, 6, 34], describing the distribution of the geometries in the reference space of D_*, are computed. In particular, two numeric values, called E_0 and E_2, derived from the box-counting ($BC^q_{D_*}(r)$) technique [3] are considered. The following formula show 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 scale, respectively.

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

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

Since every dataset has to be partitioned when running in Hadoop and Spark, we devise a fourth set of features that represent the spatial characteristics of the partitioning. Spatial partitioning is characterized by the MBR, number of geometries, and 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 4.4 (Partition statistics - P_{part}). Given a dataset D_* that is partitioned with some technique, the following parameters regarding its partitions can be defined:

- $\#cells(D_*)$: number of cells partitioning the dataset D_*.
- $\#splits(D_*)$: number of blocks containing records of D_*.
- 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., width + height.
- Total overlapping of D_*: sum of the area of the overlapping regions produced by intersecting each cell c_i with all other cells.
$$TT_{overlap}(D_*) = \sum_{c_i, c_j \in \mathcal{I}(D_*)} area(c_i \cap c_j) \times B(c_i, c_j) + \sum_{c_i \in \mathcal{I}(D_*)} \frac{c_i.blocks \times (c_i.blocks - 1)}{2}$$
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 partition information which is typically very small and can be processed by a single machine. For example, for disk-based partitioned data, this information is stored in the master file [2].

4.1.2 Combined statistics. Since spatial join is a binary operation that takes two input datasets, we include additional *combined statistics* that catch the relationship between the two inputs. The following statistics are computed for each pair of datasets that are input of a join query. Notice that we do not need an additional scan over the data to compute these statistics since they are computed based on the summaries computed for each dataset individually.

Definition 4.5 (MBR overlapping statistics - P_{mbr}). Given a pair of datasets (D_i, D_j), the following statistics describing their overlapping are computed.

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

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

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

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

- Jaccard similarity:

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

In order to combine the information stored in the high-resolution histogram computed separately on the datasets D_i and D_j, we introduce the concept of *convoluted histogram*.

The convoluted histogram is simply the result of two histograms for the two datasets overlaid on top of each other. The goal is to give the machine learning model a picture of how the two datasets overlap with each other, which is extremely important for estimating the cost of the spatial join query.

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 by the largest number to prepare it for machine learning processing.

To compute a *convoluted* histogram for two datasets, we need to apply the same grid for both of them. 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 [32]. 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 [32].

Definition 4.6 (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. 4.3. 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.

4.1.3 Performance metrics. The following metrics are collected from the result of spatial join for each pair of datasets. We propose model that trains on these collected metrics and estimates them during the inference step as further detailed in Sec. 4.4.

Join selectivity: the first performance metric is *join selectivity* (σ_{JS}), computed as:

$$\sigma_{JS} = \frac{|D_i \bowtie D_j|}{|D_i| \cdot |D_j|}$$

i.e., the cardinality of the spatial join divided by the cardinality of the cross product of the two datasets.

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

$$\sigma_{MS} = \frac{\#MBRtest(algo_i, D_i, D_j)}{|D_i| \cdot |D_j|}$$

i.e., the number of MBR tests that each specific join algorithm requires divided by the cardinality of the cross product of the two datasets. This is an important machine-independent metric that 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.

4.2 Training Data Generation

In this step, we produce the training data that will be used to build the SJML model. Our goal is to build a universal model that can work with any input so it is vital that the training data is diverse and representative of a large swath of distributions. At the same time, we need to make sure that the data generation process will not take too long. Thus, we can summarize our goals as follows:

(1) Generate representative data that can train an accurate and universal model.
(2) Reduce the overhead of generating the data.
(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 [22, 36] data generator, and publicly available real data from UCR-Star [20]. 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 we cover more cases such as when 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 in pairs to produce complex distributions, e.g., diagonal+Gaussian. In total, we have 320 synthetic datasets. This allows us to produce more than 50,000 data points that represent all pairs.

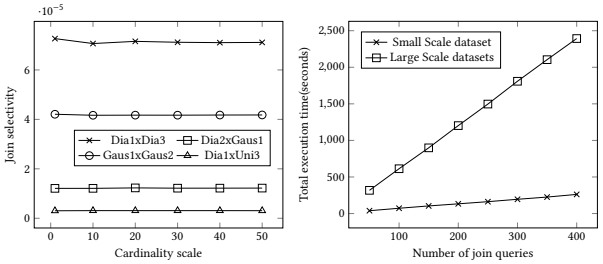
In addition to synthetic data, we also use real datasets from the public repository, UCR-Star. Since the number of real datasets is relatively limited and they are harder to retrieve and process, we generate arbitrarily many real datasets by extracting subsets of the real data with random search windows. This ensures that the distributions of the real data are diverse and representative of many real datasets. We chose the OSM buildings, lakes, and roads datasets, and US Census linearwater, edges, and faces datasets. In total, we have 95 real datasets that range from 134 MB (few HDFS blocks) to 7 GB (many HDFS blocks). This variation of number of blocks will result to the high variation of choice of the best join algorithm.

To reduce the overhead of generating the data, we generate data 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.

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 that estimates the join selectivity since it is not affected by the data size. 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 50MB. 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 Figure 3(a), the selectivity of spatial join is almost a constant which confirms this point. At the same time, as Figure 3(b) shows, joining the 1MB dataset is about 10 times faster than joining the 50MB datasets. 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 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



(a) Join selectivity (b) Join execution time

Figure 3: Join selectivity and execution time in different cardinality scales

asymptotic behavior of algorithms when running on large enough input data. Overall, our proposed query optimizer focus on HDFS-based data systems (Hadoop, Spark) and give more priority for problems on large datasets.

4.3 Training Set Preparation

In our system, we design the training data as tabular data. Each row of the training data includes a list of features and the output metric. The metric could be the join selectivity, the 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 2:

(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 Section 4.4.

Table 2: 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_{JS}	σ_{MS}
FS_s	✓	✓						
FS_{sh}	✓	✓		✓	✓			
FS_{shp}	✓	✓	✓	✓	✓	✓		
FS_{all}	✓	✓	✓	✓	✓	✓	✓	✓

4.4 Definition of the models

In this part, we describe the models that we propose for the spatial join cost estimation problem. We show how we break down the complexity of the spatial join problem and algorithms by proposing an ensemble of models. In particular, we create four models which estimate four different metrics that all help in choosing the best algorithm. The four models estimate, (1) join selectivity, (2) MBR test selectivity per algorithm, (3) the best algorithm by running time, and (4) the running time of the chosen algorithm.

4.4.1 M1: Join selectivity estimation model. This model estimates the join selectivity of the spatial join operation (σ_{JS}). There are three reasons we are considering σ_{JS} for the first model. First, it is an algorithm- and machine-independent metric which makes it relatively easier to estimate. Second, join selectivity is a very important metric for spatial join performance that is important to estimate by itself. Third, it is a metric that does not depend on the data size which allows us to use small datasets for training.

We build M1 as a regression model that takes as input the features of both inputs except for the size and the partitioning information as they are irrelevant. The learned metric join selectivity is defined as the result cardinality divided by the product of the input cardinalities. Since the join selectivity is independent of the join algorithm, we have only one instance of M1 that can be used for all join algorithms. To speed up the training process, we generate a large training set for small data and we run only one join algorithm to get the result size. We can easily parallelize this process on cluster nodes by letting each executor process one of the datasets.

4.4.2 M2: MBR test selectivity estimation model. We use MBR test selectivity (σ_{MS}) as a stable and machine-independent metric for the performance of each algorithm for two reasons. First, it is machine independent which makes it easy to port the learned model to a different system without any changes in the training. In addition, since it is system load independent, it is possible to generate the training set in parallel on a cluster without worrying about any conflicts. Second, it is independent of input size which allows us to train it on as low as medium-size data.

We train four M2 models, one for each algorithm. The model is trained as a regression model and takes as input all input features except for input size which is irrelevant for this model. Unlike M1, this model takes the partitioning features into account since the partitioning of the input affects the performance. The training set for this model consists of medium and large datasets. For medium data (10-20MB), we use a partition size of 128 KB to generate enough partitions to train this model. The resulting partitions are equivalent to partitioning a 10-20 GB dataset with a 128 MB partition size.

4.4.3 M3: Algorithm Selection. This model chooses one of the algorithms to run for a specific pair of inputs. We design M3 as a classification model that produces one of four labels for the four algorithms. Since the choice of algorithms is highly dependent on the input size, this model is only trained on large datasets with several gigabytes in size.

4.4.4 M4: Running Time Estimation. This last model estimates the end-to-end running time of a specific algorithm. This model is not necessary for the execution of SJML, but it is added to make the system more user friendly by providing an estimated running time.

Since the running time is hardware dependent, we build a linear regression model for each algorithm that is trained and applied on specific hardware settings. It takes as input the result cardinality and MBR test cardinality, estimated by models M1 and M2, and produces an estimated running time. Due to its simplicity, this algorithm can be trained on a very small training set which makes it easy to train when the algorithm is ported to a different hardware. This model is not meant to be accurate enough to rank the algorithms but can still produce a rough estimated running time.

5 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?

5.1 Experimental Setup

We implement four spatial join algorithms with their original design on Beast [10] - 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 [22, 36]. The real data is downloaded from UCR-Star [20]. We chose the OSM buildings [15], lakes [14], and roads [16] datasets, and US Census linearwater [8], edges [7], and faces [9] datasets. The detailed information of experimental datasets are described in our technical report [35].

As a baseline, we use the theoretical models proposed in [5, 21, 31]. 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.

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

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.

5.2 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 [2] and also adopted in the theoretical model for spatial join algorithm ranking proposed in [5, 31]. 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 3. 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.

Table 3: 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	$\geq$ 90	1.7%	72%
Skewed	$\geq$ 90%	25%	64%
Skewed	< 90%	$\gg$ 200%	57%

- *Baseline B2 for the MBR test selectivity model M2:* we consider the estimation of the theoretical model for ranking spatial join algorithms proposed in [5]. 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 [5] 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.
- *Baseline for algorithm selection model M3*: we compare our work with the rule-based model proposed in [31] (called $B3_1$) and the cost-based model proposed in [5] (called $B3_2$). The theoretical models were designed for uniformly distributed datasets, thus, as shown in Table 3 (column Acc_{Rank}), the accuracy of $B3_2$ in predicting at the best or the second best join algorithm decreases quickly considering non uniform datasets.

5.3 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 4: 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%**	$\mathbf{2.36}\times10^{-6}$

Table 4 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 [2] 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 5 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 6 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, Figure 4 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. 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 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 5 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.

Table 5: Effect of feature selection on MBR selectivity estimation models B2 and M2

		BNLJ		PBSM		DJ		RepJ	
Feature Set	Model	MAPE	MAE	MAPE	MAE	MAPE	MAE	MAPE	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%**	$\mathbf{3.45\times10^{-5}}$

Table 6: Effect of feature sets on algorithm selection models

Feature Set	Model	Accuracy	MAPE
FS_{shp}	$B3_1$ [31]	33.7%	122.8%
FS_s	$B3_2$ [5]	32.2%	80.5%
FS_s	M3	71%	13.9%
FS_{sh}	M3	79%	7.9%
FS_{shp}	M3	81%	7.5%
$\mathbf{FS_{all}}$	M3	**82%**	**7.4%**

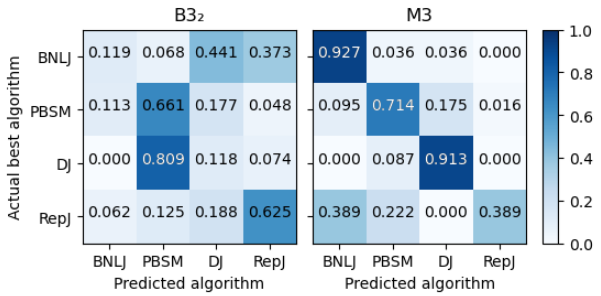


Figure 4: The confusion matrix of the algorithm selection model

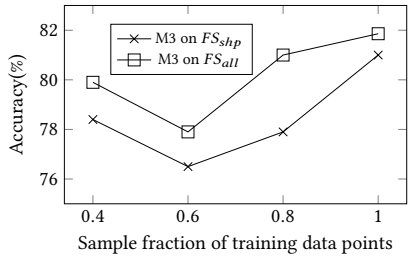
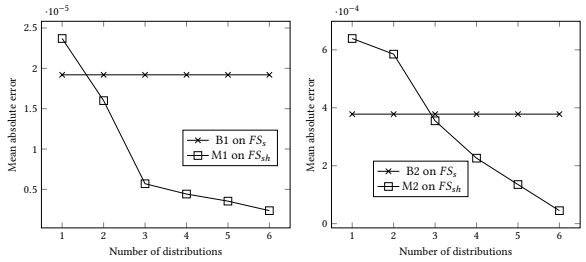


Figure 5: Performance of M3 for the FS_{shp} and FS_{all} feature sets as the training set size increases

5.4 Training Set Generation

This experiment shows the effect of the training set on the accuracy of the models. 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



(a) Join selectivity estimation model (b) MBR tests selectivity estimation model

Figure 6: Effect of training distributions in M1 and M2 model

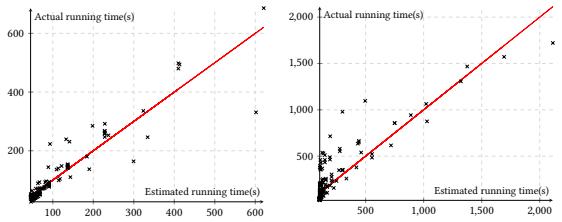
Table 7: Regression score of the linear model between join result, number of MBR tests and join execution time

Algorithm	Regression Score
BNLJ	0.86
PBSM	0.78
DJ	0.82
RepJ	0.80

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 6. These 6 distributions well represent most available spatial datasets as shown in [22, 36]. Figure 6 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 shows that we can train on synthetic data, and still estimate the join algorithm's cost efficiently.

5.5 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,



(a) BNLJ. Score: 0.864 (b) PBSM. Score:0.776

Figure 7: Spatial join running time estimation with M4

obtained by multiplying σ_{JS} and σ_{MS} (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 7 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 Figure 7. Due to limited space, we only included the models with highest score (BNLJ) and lowest score (PBSM).

6 CONCLUSION

This paper presented a spatial join cost estimation model based on machine learning. 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. Finally, the first two metrics are fed together into a regression model that estimates the running time. To train this model, 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 showed that training on synthetic and small/medium size datasets can speed up the model setting and still produce accurate results. Our experimental evaluation shows the effectiveness of the proposed method in estimating both best algorithm and join cardinality. In the future, we plan to further extend this model by taking into account the hardware specification to estimate a more accurate result.

REFERENCES

[1] Ning An, Zhen-Yu Yang, and Anand Sivasubramaniam. 2001. Selectivity Estimation for Spatial Joins. In *ICDE*. 368–375.

[2] W. Aref and H. Samet. 1994. A Cost Model for Query Optimization Using R-Trees. In *GIS*. 60–67.

[3] Alberto Belussi and Christos Faloutsos. 1998. Self-spacial Join Selectivity Estimation Using Fractal Concepts. *ACM TIS* 16, 2 (1998), 161–201.

[4] Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2018. Detecting Skewness of Big Spatial Data in SpatialHadoop *(SIGSPATIAL '18)*. 432–435.

[5] A Belussi, S Migliorini, and A Eldawy. 2020. A Cost Model for Spatial Join Operations in SpatialHadoop. *GeoInformatica* (2020).

[6] A. Belussi, S. Migliorini, and A. Eldawy. 2020. Skewness-based Partitioning in SpatialHadoop. *ISPRS IJGI* 9, 4, Article 201 (2020), 201:1 – 201:19 pages.

[7] US Census Bureau. 2019. All TIGER Lines. https://doi.org/10.6086/N1P55KJS

[8] US Census Bureau. 2019. Linear Hydrography. https://doi.org/10.6086/N1QF8QW4

[9] US Census Bureau. 2019. Topological Faces (Polygons with All Geocodes). https://doi.org/10.6086/N19021TG

[10] Ahmed Eldawy et al. 2021. Beast: Scalable Exploratory Analytics on Spatio-temporal Data. In *CIKM*. ACM.

[11] A. Eldawy and M. F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *ICDE*. 1352–1363.

[12] Ahmed Eldawy and Mohamed F. Mokbel. 2016. The Era of Big Spatial Data: A Survey. *Foundations and Trends in Databases* 6, 3-4 (2016), 163–273. https://doi.org/10.1561/1900000054

[13] Ahmed Eldawy and Mohamed F. Mokbel. 2017. Spatial Join with Hadoop. In *Encyclopedia of GIS*. 2032–2036. https://doi.org/10.1007/978-3-319-17885-1_1570

[14] A. Eldawy and M. F. Mokbel. 2019. All water areas in the world from OpenStreetMap. https://doi.org/10.6086/N1668B70

[15] Ahmed Eldawy and Mohamed F. Mokbel. 2019. The boundaries of all buildings in the world as extracted from OpenStreetMap. https://doi.org/10.6086/N1JW8BWH

[16] Ahmed Eldawy and Mohamed F. Mokbel. 2019. Roads and streets around the world each represented as a polyline extracted from OpenStreetMap. https://doi.org/10.6086/N1XK8CK6

[17] Cristian Estan and Jeffrey F. Naughton. 2006. End-biased Samples for Join Cardinality Estimation. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE*. IEEE Computer Society, 20. https://doi.org/10.1109/ICDE.2006.61

[18] Christos Faloutsos, Bernhard Seeger, Agma Traina, and Caetano Traina. 2000. Spatial Join Selectivity Using Power Laws. In *SIGMOD (SIGMOD '00)*. 177–188.

[19] Miguel Rodrigues Fornari, João Luiz Dihl Comba, and Cirano Iochpe. 2006. Query optimizer for spatial join operations. In *GIS*. ACM, 219–226.

[20] Saheli Ghosh, Tin Vu, Mehrad Amin Eskandari, and Ahmed Eldawy. 2019. UCR-STAR: The UCR Spatio-Temporal Active Repository. *SIGSPATIAL Special* 11, 2 (Dec. 2019), 34–40.

[21] Edwin H Jacox and Hanan Samet. 2007. Spatial join techniques. *ACM Transactions on Database Systems (TODS)* 32, 1 (2007), 7–es.

[22] Puloma Katiyar, Tin Vu, Sara Migliorini, Alberto Belussi, and Ahmed Eldawy. 2020. SpiderWeb: A Spatial Data Generator on the Web. In *SIGSPATIAL*. ACM.

[23] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR*.

[24] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph M. Hellerstein, and Ion Stoica. 2018. Learning to Optimize Join Queries With Deep Reinforcement Learning. *CoRR* abs/1808.03196 (2018).

[25] Viktor Leis et al. 2017. Cardinality Estimation Done Right: Index-Based Join Sampling. In *CIDR*.

[26] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Learning to Steer Query Optimizers. *SIGMOD*.

[27] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *aiDM@SIGMOD*, Rajesh Bordawekar and Oded Shmueli (Eds.). ACM, 3:1–3:4.

[28] Ryan C. Marcus et al. 2019. Neo: A Learned Query Optimizer. *PVLDB* 12, 11 (2019), 1705–1718.

[29] Kiyoshi Ono and Guy M. Lohman. 1990. Measuring the Complexity of Join Enumeration in Query Optimization. In *PVLDB*. 314–325.

[30] Fabian Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830. http://dl.acm.org/citation.cfm?id=2078195

[31] Ibrahim Sabek and Mohamed F. Mokbel. 2017. On Spatial Joins in MapReduce. In *SIGSPATIAL*. Article 21, 10 pages. https://doi.org/10.1145/3139958.3139967

[32] Samriddhi Singla and Ahmed Eldawy. 2020. Flexible Computation of Multidimensional Histograms. In *SpatialGems* (Seattle, Washington, USA). ACM.

[33] David Vengerov, Andre Cavalheiro Menck, Mohamed Zaït, and Sunil Chakkappen. 2015. Join Size Estimation Subject to Filter Conditions. *PVLDB* 8, 12 (2015).

[34] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2020. Using Deep Learning for Big Spatial Data Partitioning. *TSAS* 7, 1 (2020), 3:1–3:37.

[35] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2021. Towards a Learned Cost Model for Distributed Spatial Join: Data, Code & Models. http://www.cs.ucr.edu/~eldawy/SJML_Resources.pdf.

[36] T. Vu, S. Migliorini, A. Eldawy, and A. Belussi. 2019. Spatial Data Generators. In *1st ACM SIGSPATIAL Int. Workshop on Spatial Gems (SpatialGems 2019)*. 7.

[37] Zongheng Yang et al. 2020. NeuroCard: One Cardinality Estimator for All Tables. *PVLDB* 14, 1 (2020), 61–73.

[38] J. Yu, J. Wu, and M. Sarwat. 2015. GeoSpark: a cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL*. 70:1–70:4.