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.

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. In particular, we implement the proposed models using scikit-learn and train/test them on different datasets. To train the model, we use an open-source spatial data generator 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. 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. 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, join cost estimation, and join order enumeration. 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 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.

<table>
<thead>
<tr>
<th>Join Type</th>
<th>Non-spatial</th>
<th>Spatial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problems</td>
<td>Selectivity estimation, join cost estimation, and join order enumeration</td>
<td>Selectivity estimation and join cost estimation</td>
</tr>
<tr>
<td>Theoretical</td>
<td>[17, 25, 29, 33]</td>
<td>[1–3, 18, 19, 31]</td>
</tr>
<tr>
<td>ML</td>
<td>[23, 24, 26–28, 37]</td>
<td>This work</td>
</tr>
</tbody>
</table>

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. 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, a standard
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 pairs 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.

### 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 Single Dataset Statistics. The following statistics are computed individually for each dataset $D_i$, 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 - $P_{\text{size}}$). Given a dataset $D_i$, we define:

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

Definition 4.2 (Data density statistics - $P_{\text{dens}}$). Given a dataset $D_i$, we define:

- $\text{MBR}(D_i)$: Minimum Bounding Rectangle of $D_i$. $\text{MBR}(D_i) = (\min\{x, y, w, h\}, \max\{x, y, w, h\})$.
- $\text{mbrArea}^{\text{avg}}(D_i)$: given the MBR of geometries $r$ in $D_i$ (mbr($r$)), it represents the average area of such MBRs. $\text{mbrArea}^{\text{avg}}(D_i) = \frac{\sum_{r \in D_i} \text{area}((\text{mbr}(r)))}{|D_i|}$.
- $\text{len}^{\text{avg}}_x(D_i)$ and $\text{len}^{\text{avg}}_y(D_i)$: given the MBR of all geometries in a dataset, they represent the average length on the $X$ and $Y$ axis of such MBRs. $\text{len}^{\text{avg}}(D_i) = \frac{\sum_{r \in D_i} \text{len}(\text{mbr}(r))}{|D_i|}$.
- $\text{len}^{\text{avg}}_x(D_i)$ 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 $8192 \times 8192$. 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 $H(D_i)$ that will be used for the computation of the convoluted histogram described shortly.

Definition 4.3 (Histogram-based statistics - $P_{\text{dist}}$). Given a dataset $D_i$, 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 $D_i$, are computed. In particular, two numeric values, called $E_0$ and $E_2$, derived from the box-counting ($BC^q_{D_i}(r)$) 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 $BC^0_{D_i}(r)$ and $BC^2_{D_i}(r)$ in log scale, respectively.

$$BC^q_{D_i}(r) = \sum_{i} h_i(D_i, r)^q$$

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

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 - $P_{\text{part}}$). Given a dataset $D_i$ that is partitioned with some technique, the following parameters regarding its partitions can be defined:

- $\#\text{cells}(D_i)$: number of cells partitioning the dataset $D_i$.
- $\#\text{blocks}(D_i)$: number of blocks containing records of $D_i$.
- $\text{total area of } D_i$: sum of the area of all partitions.

$$\text{area}(c_i) \times \#\text{blocks}(c_i)$$

where $I(D_i)$ is the set of cells used for partitioning the reference space of $D_i$ and $\#\text{blocks}(c_i)$ is the number of blocks stored in cell $c_i$.

- $\text{total margin of } D_i$: sum of the length of the semiperimeter of all partitions.

$$\text{sp}(c_i) \times \#\text{blocks}(c_i)$$

where $\text{sp}(c_i)$ is the semiperimeter of the cell $c_i$, i.e., width + height.

- $\text{total overlapping of } D_i$: sum of the area of the overlapping regions produced by intersecting each cell $c_i$ with all other cells.

$$\text{area}(c_i \cap c_j) \times B(c_i, c_j) + \sum_{c_i \in I(D_i)} \frac{\#\text{blocks}(c_i)}{2} \left(\text{area}(c_i) \times \text{sp}(c_i) - 1\right)$$

where $B(c_i, c_j) = \#\text{blocks}(c_i) \times \#\text{blocks}(c_j)$.

- Load balancing: standard deviation of index cells cardinality

$$\text{LB}(D_i) = \sqrt{\sum_{c_i \in I(D_i)} (\#\text{card}(c_i) - \text{avgCard})^2 \over |I(D_i)|} \text{avgCard}$$

where $\text{avgCard}$ is the average cardinality of the cells belonging to the index $I(D_i)$.

- Block utilization: average percentage of block usage:

$$\text{BU}(D_i) = \sum_{c_i \in I(D_i)} \text{blocks}(c) \times \text{size}(c) \times \text{card}(c) \over |I(D_i)| \times \text{blocks}(c)$$

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.
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 dataset 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 are explained in [32].
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 linearmr, 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

![Figure 3: Join selectivity and execution time in different cardinality scales](image)

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>
<thead>
<tr>
<th>Name</th>
<th>Features of $D_s$</th>
<th>Features of $(D_i, D_j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FS_S$</td>
<td>✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>$FS_{sh}$</td>
<td>✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>$FS_{shp}$</td>
<td>✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>$FS_{all}$</td>
<td>✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
</tbody>
</table>

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 ($\sigma_{JS}$). There are three reasons we are considering $\sigma_{JS}$ 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 ($\sigma_{MBR}$) 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 2x8-core Intel Xeon CPU E5-2609 v4 @1.7GHz, and 12 executor nodes each with 2x6-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.

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

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 datasets (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, B2) for join selectivity and the best two algorithms in ranking.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>MBR overlap</th>
<th>MAPEFS (B1)</th>
<th>Acc (B32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>≥ 90</td>
<td>1.7%</td>
<td>72%</td>
</tr>
<tr>
<td>Skewed</td>
<td>≥ 90%</td>
<td>25%</td>
<td>64%</td>
</tr>
<tr>
<td>Skewed</td>
<td>&lt; 90%</td>
<td>≫ 200%</td>
<td>57%</td>
</tr>
</tbody>
</table>

- **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 B31) and the cost-based model proposed in [5] (called B32). The theoretical models were designed for uniformly distributed datasets, thus, as shown in Table 3 (column AccRank), the accuracy of B32 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 FSx, FSsh, and FSshp. 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

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>Model</th>
<th>MAPE</th>
<th>MAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSx</td>
<td>B1</td>
<td>35%</td>
<td>2.87\times10^{-3}</td>
</tr>
<tr>
<td>FSx</td>
<td>M1</td>
<td>23%</td>
<td>1.06\times10^{-3}</td>
</tr>
<tr>
<td>FSsh</td>
<td>M1</td>
<td>4.49%</td>
<td>2.36\times10^{-6}</td>
</tr>
</tbody>
</table>

Table 4 shows the evaluation of join selectivity model (M1) and the baseline (B1) when using features sets FSx and FSsh. We do not use FSshp 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 FSx. 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 FSx. Given the same feature set FSx, a random forest regression model M1 is significantly better than the baseline B1 gaining 12% in terms of MAPE. Finally, if we feed FSsh 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 FSx, FSsh, and FSshp. 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 down-trend 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 (B31 and B32) 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 B31 and B32 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 B32 and the proposed M3, respectively. This figure shows that the baseline get a decent accuracy (around 60%) for the PBSM and RepJ. 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 FSshp and FSall. 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 FSall even when the training set is small. This indicates that the join selectivity and MBR selectivity, which are added in FSall, help in improving the model performance.
Table 5: Effect of feature selection on MBR selectivity estimation models B2 and M2

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>Model</th>
<th>MAPE</th>
<th>MAE</th>
<th>MAPE</th>
<th>MAE</th>
<th>MAPE</th>
<th>MAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>𝐹𝐒ₛ</td>
<td>B2</td>
<td>518%</td>
<td>1.13×10⁻²</td>
<td>1332%</td>
<td>7.18×10⁻⁴</td>
<td>386%</td>
<td>4.36×10⁻⁴</td>
</tr>
<tr>
<td>𝐹𝐒ₛ</td>
<td>M2</td>
<td>15%</td>
<td>6.64×10⁻⁴</td>
<td>1.25%</td>
<td>3.99×10⁻⁵</td>
<td>6.8%</td>
<td>11.09×10⁻⁵</td>
</tr>
<tr>
<td>𝐹𝐒ính</td>
<td>M2</td>
<td>1.8%</td>
<td>6.13×10⁻⁵</td>
<td>1.42%</td>
<td>4.48×10⁻⁵</td>
<td>5.4%</td>
<td>8.25×10⁻⁵</td>
</tr>
<tr>
<td>𝐹𝐒ính</td>
<td>M2</td>
<td>1.77%</td>
<td>6.22×10⁻⁵</td>
<td>1.39%</td>
<td>4.38×10⁻⁵</td>
<td>4.5%</td>
<td>6.95×10⁻⁵</td>
</tr>
</tbody>
</table>

Table 6: Effect of feature sets on algorithm selection models

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>Model</th>
<th>Accuracy</th>
<th>MAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>𝐹𝐒ₛ</td>
<td>B3₁ [31]</td>
<td>33.7%</td>
<td>122.8%</td>
</tr>
<tr>
<td>𝐹𝐒ₛ</td>
<td>B3₂ [5]</td>
<td>32.2%</td>
<td>80.5%</td>
</tr>
<tr>
<td>𝐹𝐒ₛ</td>
<td>M3</td>
<td>71%</td>
<td>13.9%</td>
</tr>
<tr>
<td>𝐹𝐒ính</td>
<td>M3</td>
<td>79%</td>
<td>7.9%</td>
</tr>
<tr>
<td>𝐹𝐒ính</td>
<td>M3</td>
<td>81%</td>
<td>7.5%</td>
</tr>
<tr>
<td>𝐹𝐒all</td>
<td>M3</td>
<td>82%</td>
<td>7.4%</td>
</tr>
</tbody>
</table>

Figure 4: The confusion matrix of the algorithm selection model

Figure 5: Performance of M3 for the 𝐹𝐒ính and 𝐹𝐒all feature sets as the training set size increases

5.4 Training Set Generation

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Regression Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNLJ</td>
<td>0.86</td>
</tr>
<tr>
<td>PBSM</td>
<td>0.78</td>
</tr>
<tr>
<td>DJ</td>
<td>0.82</td>
</tr>
<tr>
<td>RepJ</td>
<td>0.80</td>
</tr>
</tbody>
</table>

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

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,
obtained by multiplying \( \sigma JS \) and \( \sigma MS \) (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