Towards a Learned Cost Model for Distributed Spatial Join: Data, Code & Models*

Tin Vu
Microsoft Corporation
Redmond, WA, USA
khactinvu@microsoft.com

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

Geospatial data comprise around 60% of all the publicly available data. One of the essential and most complex operations that brings together multiple geospatial datasets is the spatial join operation. Due to its complexity, there is a lot of partitioning techniques and parallel algorithms for the spatial join problem. This leads to a complex query optimization problem: which algorithm to use for a given pair of input datasets that we want to join? With the rise of machine learning, there is a promise in addressing this problem with the use of various learned models. However, one of the concerns is the lack of standard and publicly available data to train and test on, as well as the lack of accessible baseline models. This resource paper helps the research community solve this problem by providing synthetic and real datasets for spatial join, source code for constructing more datasets, and several baseline solutions that researchers can further extend and compare to.

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. 2022. Towards a Learned Cost Model for Distributed Spatial Join: Data, Code & Models. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3511808.3557712

*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

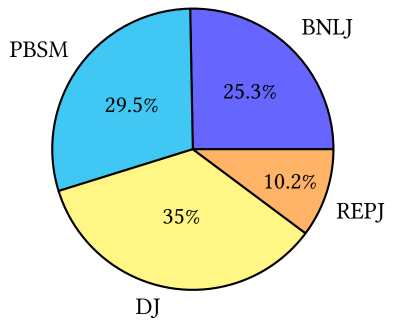


Figure 1: Distribution of best join algorithm in terms of running time

1 INTRODUCTION

In the era of data science, real data is taking part of all scientific projects. The majority of these projects require some kind of geospatial data which provide contextual information about the whereabouts of the study. Spatial join is the main operation that combines multiple datasets based on the geolocation. For example, it can find geographically overlapping records or run a point-in-polygon query to associate GPS points with regions of interest. The importance of spatial join led to a large number of spatial join algorithms for big data [2–4, 6, 8, 13, 14].

This large number of solutions for executing a spatial join raises several challenging research problems in query processing and query optimization. In this resource paper, we focus on the problem of spatial join query optimization, that is, given two input datasets that a user wants to join, how to estimate the cost of spatial join and how to choose the most appropriate algorithm? In particular, 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). Unfortunately, the complexity of the spatial join and the modern big-data systems make these problems extremely difficult. For example, Figures 1 shows the distribution of the best join algorithm in running time when we execute them on 300 pairs of different datasets. This distribution indicates that it is challenging to choose an appropriate algorithm for a random join input. This motivated our research to investigate the insights of distributed spatial join algorithms.

The rise of machine learning makes it an attractive solution to break the complexity of estimating the cost of spatial join. However, to obtain a good model for such a complex problem, we need a training set of a large size. The training set must be diverse enough to catch most of the aspects of spatial join. However, this is a time consuming and expensive process due to the complexity of spatial join. Additionally, machine learning models need both good and bad examples equally, so we need to run a lot of inefficient algorithms to make sure that the model will learn to avoid them in specific scenarios. Each data point, which is corresponding to a join operation, could take few hours to complete. For example, to generate a small training set of 300 data points, we had to occupy a 12-node cluster for 190 hours which is an expensive process that not everyone has access to.

In this paper, we publish a detailed and curated dataset that has been carefully designed and curated for over a year and would take at least a month to reproduce, if at all possible, for other researchers. We generated this resource in the course of building a machine learning model for spatial join. Interested readers can find more details on our recently published paper [11]. The dataset has more than 7,000 data points and has been collected from two in-house clusters at UC Riverside and University of Verona, and a cloud-based cluster hosted by Amazon Web Services (AWS). We also include all the code and scripts that allow researchers to generate similar datasets with similar or different characteristics. Finally, we publish a first-cut machine learning model and a theoretical model that can be used as baselines for comparison.

The rest of this paper is organized as follows. Section 2 gives the details of the datasets and proposed features included in this resource. Section 3 illustrates the spatial join cost models that we make available as part of this resource. Section 4 gives a quick overview on how to access and extend the resource. Finally, Section 5 concludes the paper and gives some future directions.

2 DATASETS AND PROPOSED FEATURES

In this section, we describe the available datasets in this resource that can be used to devise spatial join cost models. The described approach is fully reproducible and extensible. Therefore, researchers can follow this process to generate their own datasets if they have specific requirements. Otherwise, researchers can directly use our published datasets, which are already suitable for different purposes. In particular, our published datasets include both synthetic and real data [10]. The synthetic datasets includes the data in different distributions and scale. The real datasets are provided to validate that our proposed models work well on both synthetic and real datasets. Moreover, we also provide a dataset which contains hand-crafted features, which were proven to be useful for prediction problems. Researchers can directly use these proposed features, or extend them with other features based on their own problems.

2.1 Synthetic datasets

The first step to build our learned models for spatial join cost estimation is to generate or collect spatial datasets with variable characteristics. In particular, we utilized an open-source spatial data generator [7, 12] to generate synthetic datasets. Overall, the spatial data generator requires the following parameters to generate a dataset: cardinality, distribution, number of dimensions, type of geometries, and output format (CSV or GeoJSON). Users have several options to generate their dataset using the spatial data generator. First, the most convenient method is to generate data using Spider [7], an interactive web-based generator for spatial datasets. Spider provides the visualization of user's generated datasets. The web interface is intuitive and easy to use, but might be difficult to generate batch of datasets. This is the reason we also provide the Python and Spark API, in which user can programmatically generate their datasets. If the scale of dataset is not too large, users can use the published Python program [12] to generate a spatial dataset as in the following example:

```
python3 generator.py distribution=uniform
 cardinality=100 dimensions=2 geometry=point
```

If users want to generate large-scale datasets and store them in HDFS, our Spark API would be the best option to complete this requirement.

```
import edu.ucr.cs.bdlab.beast._
import edu.ucr.cs.bdlab.beast.generator._
val generatedData: SpatialRDD =
 sparkContext.generateSpatialData(
  UniformDistribution, 100,
  opts=Seq(SpatialGenerator.Dimensions->2,
  PointBasedGenerator.GeometryType->"point"))
```

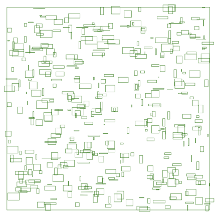
The open-source Python program and Spark API allow users to generate spatial datasets in batch. For example, they can define the required parameters in a CSV file, then read each row of that CSV file to generate the corresponding dataset. In our own datasets for the learned spatial join cost model, we generate datasets in different scale: small, medium and large and different distributions: uniform, diagonal, gaussian, bit, parcel, as shown in Figure 2. In addition, we also add several real datasets to verify that our model could be trained using synthetic data and the trained model will still work well on real data. The detail will be discussed in Section 2.2.

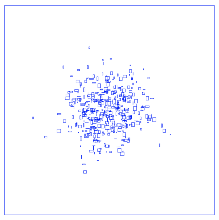
2.2 Real datasets

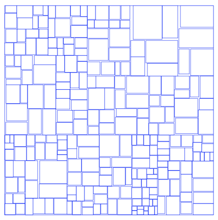
While the spatial data generator that we used provide huge flexibility with data distributions and parameters, it cannot fully replace real datasets with real distributions. Therefore, we provide a group of real datasets as part of this resource. To avoid adding tens of real datasets that might have similar distributions, we created 82 real datasets by spatially partitioning several large-scale datasets, as illustrated in Figure 3. We take the publicly available datasets and use a grid to spatially partition each of them into smaller datasets, each representing a different geographical region with different characteristics. This approach is better than using the full real datasets since it can be used to create arbitrarily many datasets from a single dataset. In this resource, we propose the script that partitions a real dataset into smaller ones as well as the partitioned datasets illustrated in Figure 3. All these real datasets are downloaded from UCR-Star [5] and users can download more datasets and partition them in the same way.

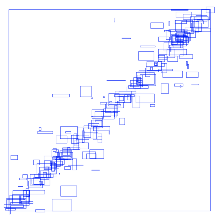
2.3 Proposed features and performance metrics

The datasets described in Sec 2.1 and 2.2 are already useful for many researchers in different purposes. In this section, we take one step further to provide another option for researchers who


(a) Uniform


(b) Gaussian


(c) Parcel


(d) Diagonal

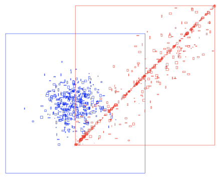
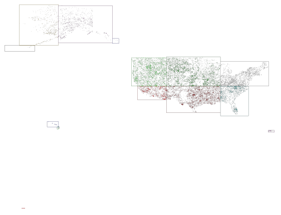
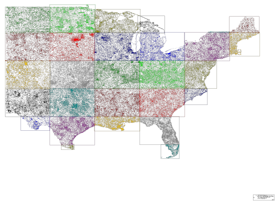

(e) Combo

Figure 2: Some of the synthetic datasets included in this resource


(a) TIGER2017 LINEARWATER


(b) TIGER2018 Faces

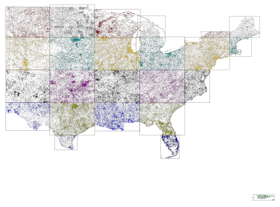

(c) TIGER2018 Edges

Figure 3: Partitioned real datasets. Each of the six main datasets is further partitioned to create multiple smaller real datasets

want to build machine learning models on insightful features from spatial join inputs. This is a reasonable requirement, since most of machine learning models expect fixed-size feature vectors for the training/testing process. Given the spatial join inputs D_i and D_j, we compute a set of features that effectively reflect the knowledge that is necessary to the model for estimating the cost of a join operation $D_i \bowtie D_j$. The cost metric could be join selectivity, number of MBR tests selectivity, or the best join algorithm in terms of running time. In Table 1 we summarize the proposed set of features in different groups. Interested readers can find more details on our published paper [11].

Since spatial join is an expensive operation, collecting data from a huge amount of join queries is challenging and time-consuming. Therefore, this dataset would be helpful for researchers who want to study the insights of spatial join queries, but do not have enough time or hardware resources to execute their own queries. To generate the spatial join execution dataset, we make a list of join pairs, each pair includes two input datasets. Then we execute four join algorithms, including BNLJ, PBSM, DJ and REPJ, for each pair of datasets. Once this process is complete, we can collect several metrics such as join cardinality, the number of MBR tests and the running time for each join algorithm.

2.4 Datasets for spatial join cost models

This section discusses the dataset which would be used to train/test/evaluate in the spatial join cost models. This is a tabular dataset in which each row contains the extracted features and performance metrics discussed in Section 2.3. In particular, we computed total 35 features as shown in Table 1. The dataset also include three performance metrics which are join selectivity, MBR tests selectivity and the best join algorithm. Thus, researchers can use this dataset for different kinds of spatial join cost model, based on their specific

Table 1: Summary of features

	Computation	Group	Feature	Description
Single dataset D	File system	Size	$\#geo$	Number of geometries
			$size$	Total input size
	Scan(D)	Density	mbr	Minimum bounding rectangle covered by D
			$area^{avg}$	Average record area
			len_x^{avg}	Average record width
			len_y^{avg}	Average record height
	`Histogram`(D)	Distr.	E_0, E_2	Box counting with base 0 and 2
	`MasterFile`(D)	Partitioning & indexing	$\#cells$	Number of cells (partitions)
			$\#splits$	Number of splits (blocks)
			$area_{cells}^{sum}$	Sum of partition areas
			$margin_{cells}^{sum}$	Sum of partition semi-perimeters
			$overlap_{cells}^{sum}$	Sum of overlap areas between pairs of partitions
			LB_{blocks}	Load balancing: Standard deviation of block sizes
			BU_{blocks}	Block utilization: Percentage of block usage
Pair of datasets	`Overlap`(mbr_i, mbr_j)	Overlap	$Area_i$ $Area_j$	Percentage of overlap area occupied by dataset i or j
			$AreaInt$	Overlap area
			$JacS_{i,j}$	Jaccard similarity between the two MBRs
	`Conv`(h_i, h_j)	$\cap$ distr.	e_0, e_2	Box counting for the convoluted histogram with base 0 and 2

requirements. We published this dataset [10] so that researchers can directly feed the data to their cost models. In case user want to generate the features of their own datasets, they can utilize the programs in our open-source project [10] to compute the same features and metrics from their join queries.

3 PRE-TRAINED MODELS

In this section, we describe our publicly available pre-trained models for spatial join optimization problems. In particular, we provide both proposed machine learning based models and baseline models for comparison purpose. Researchers can utilize our pretrained models in their query optimizers, or use them as the baseline to compare with their own proposed cost models.

Join selectivity estimation models: *Join selectivity (JS)* is defined as the ratio between the actual number of pairs produced by the join operation and the total number of pairs produced by the cross product. When *JS* is estimated, the corresponding join cardinality can be obtained by multiplying it for the size of the cross product. We provide the implementation for both baseline model [1] and proposed machine learning model [11]. Our proposed model can achieve up to 4.49% of the mean absolute percentage error (MAPE), which is pretty good when compared to the 35% error of the theoretical formula.

MBR tests selectivity models: *MBR tests selectivity* is the ratio between the number of MBR tests executed by the join operation and the total number of pairs produced by the cross product. Since each join algorithm has a specific strategy, thus the number of MBR tests is a algorithm-dependent metrics. Based on this fact, we build four separate models to predict the MBR tests selectivity for four spatial join algorithms (BNLJ, PBSM, DJ, REPJ). Each model is a random forest regressor that takes the extracted features as the input, then predict the MBR tests selectivity. Our experiments shows that the proposed models can achieve up to 1.77%, 1.25%, 4.5%, 3.3% MAPE value for BNLJ, PBSM, DJ, and REPJ, respectively. These performance is much better when compared to the theoretical formula above. The detailed experimental results can be found at our published paper [11].

Algorithm selection models: The baseline algorithm selection models are implemented based on the ideas in [9] and [2]. The proposed model is a random forest classification model that take the extracted features as the input, then predict the best spatial join algorithm in terms of running time. We train and test the algorithm selection models using the dataset described in Section 2.4. Our published paper [11] shows that the proposed machine learning model outperformed other rule-based algorithm and theoretical model in the algorithm selection problem.

4 RESOURCE GUIDE

In this section, we briefly describe a guideline of how to access our published resources, including the datasets, source code and models. More importantly, we also show how to extend our provided resources to create your own datasets and models.

4.1 Download datasets and run the models

We publish all of our datasets and source code in a Github repository [10]. Researchers who are interested in these datasets could download them and use directly, or they can also create their own datasets using the web interface, Python program or Spark API. Researchers can easily execute our code without any further installation. Once the datasets are ready, users can use them in their own spatial join cost models, or just train/test them on our proposed models. The train/test process are simple and intuitive. For example, users can run the simple commands in the tutorial to train and test a random forest classification model that predicts the best join algorithm. Along with the algorithm selection model, our published source code also includes the regression models to predict join selectivity, MBR tests selectivity with both random forest and decision tree as the estimator.

```
python main.py --model clf_random_forest --tab
 tabular_test_data_path.csv --path model_path.h5
--weights model_weights_path.h5 --no-train
```

4.2 Extend current works

Although our published resources already provide diverse insights about distributed spatial join operation, it is a reasonable need if researchers want to extend our resources to build their own models. The source code is also easy to extend with minimal effort if users want to run the models on other datasets with different features, or run the model with different core algorithms (linear models, SVM, SGD, Nearest Neighbors, Naive Bayes, etc). In addition, we provide data of high resolution histograms of input datasets, which can be used in deep learning based model. The reason is that histogram data could carry hidden characteristics, which might tell us some useful information of the join query. This would be a good direction to extend our work to have a better prediction accuracy and reduce the works of hand-crafted feature extraction.

5 CONCLUSION AND FUTURE WORK

This paper presented a publicly available resource that will facilitate new research directions in improving spatial join query processing and optimization. The resource consists of three parts, datasets, feature extraction, and baselines. In datasets, we provide both synthetic and real datasets that cover a wide range of distributions. These datasets can be used for benchmarking spatial join. We also provide the source code that researchers can use and extend to generate more datasets following the same idea. Second, for feature extraction, we provide a set of standard features that can be exploited by machine learning to build query optimization models for estimating the cost of spatial join or to select the best algorithm. We provide both the source code that calculates the features and precalculated features for all the synthetic and real datasets that we provide. Third, we provide theoretical and machine learning cost models for estimating the size of the spatial join and the cost of the algorithm. These models can be used as baselines for evaluating new cost models to these standard models. We envision that this resource will help more researchers in the information management area to build new standardized query optimizers for spatial join and related queries.

REFERENCES

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

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

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

[4] 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

[5] 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.

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

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

[8] Suprio Ray, Bogdan Simion, Angela Demke Brown, and Ryan Johnson. 2014. Skew-resistant parallel in-memory spatial join. In *SSDBM*. ACM, Aalborg, Denmark, 6:1–6:12.

[9] 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

[10] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2021. Github repository for paper's data, code and models. https://github.com/tinvukhac/sjml-resources.

[11] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2021. A Learned Query Optimizer for Spatial Join. In *Proceedings of the 29th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 458–467.

[12] 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.

[13] Simin You, Jianting Zhang, and Le Gruenwald. 2015. Large-scale spatial join query processing in cloud. In *2015 31st IEEE international conference on data engineering workshops*. IEEE, 34–41.

[14] 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.