Spatial Embedding: A Generic Machine Learning Model for Spatial Query Optimization*

Alberto Belussi
Sara Migliorini
Dept. of Computer Science
University of Verona, Italy
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

Machine learning and deep learning techniques are increasingly applied to produce efficient query optimizers, in particular in regards to big data systems. The optimization of spatial operations is even more challenging due to the inherent complexity of such kind of operations, like spatial join, range queries, and the peculiarities of spatial data. Even though a few ML-based spatial query optimizers have been proposed in literature, their design limits their use, since each one is tailored for a specific collection of datasets, a specific operation, or specific a hardware. Changes to any of these will require building and training a completely new model which entails collecting a new very large training data to obtain a good model

This paper proposes a new approach for ML-based query optimization which exploits the use of the novel notion of *spatial embedding* for overcoming these limitations. In particular, a preliminary model is defined which captures the relevant features of spatial datasets, independently from the operation to be optimized and in an unsupervised manner. Given that, a specialized model for the optimization of each spatial operation can be trained by using spatial embeddings as input, so the cost of building the first model can be amortized and a smaller training set is required for the specialized ones.

CCS CONCEPTS

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

KEYWORDS

query optimizer, machine learning, big data, range query

ACM Reference Format:
Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2022. Spatial Embedding: A Generic Machine Learning Model for Spatial Query Optimization. In *The 30th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '22), November 1–4, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3557915.3560960

*This work is supported in part by the National Science Foundation, grant IIS-2046236

SIGSPATIAL '22, November 1–4, 2022, Seattle, WA, USA

ACM ISBN 978-1-4503-9529-8/22/11.
https://doi.org/10.1145/3557915.3560960

1 INTRODUCTION

In the last years, big data analytics has increased its strategic role in supporting decision systems, thanks to the growing availability of data, sometimes in heterogeneous formats. Very often georeferenced data and spatial objects represent a significant part of the datasets subject to analysis, leading to the development of many spatial big data systems and libraries [5, 6, 20]. Due to the complexity and richness of some kinds of analysis, in many cases data processing tasks are structured as pipelines of operations, and this allows the definition of many alternative road maps to produce the requested result. Moreover, the existence of different alternatives could also be originated by the fact that each single operation on spatial data can be implemented in several ways and applying different algorithms. This fact is further exacerbated in distributed systems, since the parallel execution of an algorithm often requires the tuning of several parameters, which depend on both the system configuration and the characteristics of datasets at hand. The immediate consequence of such richness of alternatives is the increasing importance covered by *query optimizers* able to automatically select the best execution plan in terms of performances.

The spatial query optimization problem could be formulated as follows: given an operation OP to be performed, the input dataset (or datasets) and the available hardware and software cluster configuration: (i) identify the different alternative implementations of OP available on the cluster, (ii) estimate the cost of each of them and choose the best one OP_i based on the characteristics of the input dataset (or datasets), and (iii) estimate the best parameter configurations for tuning OP_i in the given cluster. The problem becomes even more complicated if the data processing requires a sequence of operations to be applied (like in a pipeline), since the optimization of each single operation could also depend on the operations previously executed or on the operation order. In general, the main objective of each optimization strategy is to produce an estimation of the cost of each available alternative solution.

Recently several solutions have been proposed in literature to address the optimization problem described above [1, 3, 4, 7–9, 11–13, 17, 19]. In order to provide an effective solution to this complex task, many of them rely on estimation techniques that are based on machine learning or deep models. However, different and completely independent models are proposed, one for each estimation to be produced, even though they could share a lot of similarity. For example, two models that estimate the range query selectivity and range query running time are expected to share some similarity. As a result, each model requires a heavy-weight training step that works on a large training set, since each model is

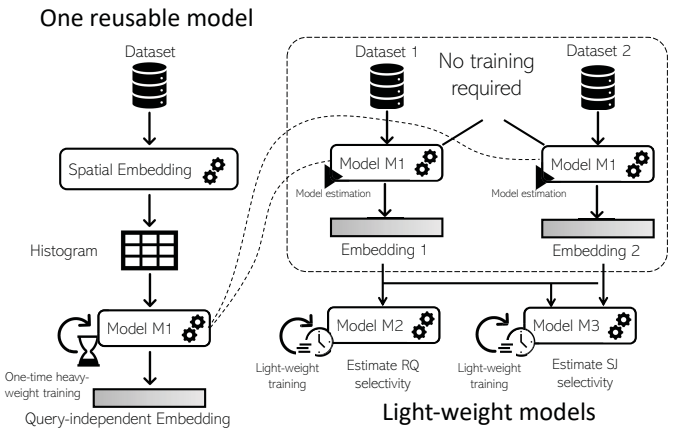

Figure 1: The proposed approach trains a model for spatial embedding and reuses it in several models to reduce training time.

designed to capture the intricate relationship between input data, query characteristics, and hardware specifications.

Given all these challenges, the main limitations of existing solutions can be summarized as follows: (i) Each model is tailored to one specific collection of datasets. Thus, if this set changes significantly, the model must be retrained, consistently reducing the generality and reusability of the solution. (ii) They are usually dedicated to a single specific operation. Thus, the extension to other operations requires to collect new data points, namely to execute a huge number of experiments, and subsequently to retrain the model on them. Finally, (iii) the results obtained with a specific cluster cannot be easily generalized to different clusters, since also the system configuration has an impact on the cost estimation. Therefore, such expensive activity, performed for building the training set, should be repeated for each cluster configuration. Some attempts to overcome this limitation have been proposed in [16] where some metrics have been proposed for determining the best partitioning technique, which are independent from the adopted cluster configurations and depend only on the dataset features. Similarly, some metrics independent from the cluster configurations are adopted also in [17] where some models for estimating the fastest implementation of a spatial operation have been proposed. However, this last proposal leads to another limitation: (iv) these solutions are based on the extraction of some features from the input datasets and such features can be different according to the operation we need to estimate the cost of. Thus, they must be recomputed for each operation we consider.

This paper proposes a general ML-based model for spatial query optimization that overcomes the limitations of existing work. It can estimate *different cost parameters* regarding the execution of *multiple implementations* of some *spatial operations* which are independent from the specific input datasets and the cluster characteristics. The proposed framework, illustrated in Fig. 1, is characterized by the following components: (1) A first machine learning model ($M1$) that is trained on a large collection of spatial datasets with the goal of extracting a set of significant features that are *independent of the operation and the hardware*. Since this model is independent, it needs to be trained only once and can be reused for any spatial operation that we want to build an estimation model for. Therefore, we can invest more time in it. We call this model *spatial embedding distiller* since it creates a compact summary of the spatial dataset that can then be used in any specific model. To build the spatial embedding of a dataset D, we generate and use a *multifaceted histogram* representing the distribution of some features in the reference space of D (Minimum Bounding Rectangle of D). Furthermore, model $M1$ can be trained in an *unsupervised manner* with a huge amount of datasets, even automatically generated with tools like [7, 18], and whose result does not depend on the specific spatial operation.

(2) Given an operation and a specific implementation for it, a set of *cost parameters* are chosen. Operations can be for example: range query (RQ), spatial join (SJ), of k-NN (KNN). Different implementations of the same operation can be: index-based and scan-based RQ algorithm, or partition-based and index-based SJ algorithm.

(3) For each chosen combination of an operation o, implementation i, and parameters p, a model $M2(o, i, p)$ is trained starting from the spatial embeddings of the input dataset(s) produced by the first model. Since M1 is already trained, the size of data required for the training of M2 is reduced, consequently reducing its training cost.

In summary, as illustrated in Fig. 1, the proposed framework is composed of an unsupervised model $M1$ for producing spatial embeddings, and a supervised model $M2$ for each operation we want to build a machine learning for. Such framework requires the generation of the following sets of data points: For model $M1$, a collection of synthetic datasets covering the most common distribution of real spatial data can be obtained by using the Spider tool [7]. For model $M2(o, i, p)$, each data point corresponds to an execution of the chosen implementation i of the operation o where the cost parameter p is measured. Thus, the generation of such data points is costly and can require several hours of processing, depending on the operation we consider.

2 SPATIAL EMBEDDING

Deep learning techniques often follow a pattern that tries to organize the architecture of a neural network in two steps. In the first step the goal is to extract the significant information that is contained in the input data so that the second step can be fed with a distillate of the original input where noise has already been purged. This condensed information is usually called *embedding* and it can be orders of magnitude smaller than the original data. The second step focuses on the prediction of the target value, which can be a class of a taxonomy or a forecast of a given parameter. The distillation of the embedding is very often applied in natural language processing [14] and image processing [10], with the additional effect that the next step is proven to gain accuracy with respect to models in which embeddings are not distilled.

A neural network used for this kind of task is called *autoencorders* and can be implemented as a stack of fully connected layers (*stacked autoencoders*), or of convolutional neural layers (*convolutional autoencoders*). In a stacked autoencoder, each layer has the responsibility to reduce the dimensionality of the input with also the aim to detect at each step the most interesting features. Conversely, a convolutional autoencoder typically reduces the spatial

dimensionality of the inputs (i.e., height and width) while increasing the depth (i.e., the number of feature maps). A convolutional autoencoder has typically a flatten layer which reshapes the output in order to produce the final embedding.

In its general form, an autoencoder has a symmetric architecture, namely it includes also a stack of reverse layers which allow to reconstruct the original input data set starting from the obtained embedding. In this way, the training of an autoencoder *AE* is unsupervised, since the input set of data points is at the same time the training set and the ground truth for the training.

In the paper we propose to apply this approach to the spatial optimization problem; in particular, for generating a condensed representation of the input dataset which correctly synthesizes its peculiar characteristics w.r.t. the cost of spatial operations. Since this distillation process regards a generic spatial dataset, represented as a collection of vector geometries, instead of an image, it is necessary to perform the following preliminary operations: (1) Define a format for the input of the model, indeed while images have a fixed structure (i.e., a grid of pixels), spatial dataset can vary a lot. (2) Generate a huge number of spatial datasets that cover as much as possible of the different real distributions that characterize geographical information on the Earth surface. (3) Define the structure of the neural network that, after training, can be used for generating the spatial embedding.

Regarding point (1), we choose to compute for each dataset a *multifaceted histogram* of fixed size, as done in many previous work on the field [15]. To decide which features to store in each cell of the histogram, we consider the goal of the successive models: *evaluate the cost of an operation on spatial data*. Usually such cost is influenced by: the **dataset size**: measured by cardinality (f_1) and size in bytes (f_2) of the input, the **complexity of the geometries** contained in the dataset: measured by area of the geometries (f_3), length on x and y axes of their MBR (f_4 and f_5), and number of vertices (f_6) of their vector representation. These features are computed in each cell of the histogram so that the distribution of the dataset "complexity" in the reference space can be represented by the histogram itself. In particular, in each cell c we compute the sum of features f_1, f_2 and f_6 and the average for features f_3, f_4 and f_5 considering only the geometries that intersect the cell c.

Conversely, for point (2), we generate a collection of about 2,700 synthetic datasets with different distributions (1145 of uniform, 345 of diagonal, 343 of Gaussian, 225 of parcel, 305 of bit, and 200 of Sierpinski distribution) by using the Spider tool [7]. We also choose different sizes from a minimum of about 1MB to more that 2GB. Notice that the number of datasets with uniform distribution is higher with respect to the number of other distributions. This choice is necessary because with highly skewed distributions there are many empty cells inside the reference space. Therefore, to ensure that the computed histograms are balanced between zero and non-zero values, and to avoid creating a model that over-estimates zeros, we need in this phase a bigger number of uniform distributed datasets which compensates the overall presence of skewed distributions (independently from their type).

As final preparation step, we need to normalize the values of the features in each cell of the histograms. Different functions can be applied to normalize data before training. We chose a min-max scaler, which requires firstly to compute the minimum and the maximum value for each feature among all histograms, producing two arrays of values $min[]$ and $max[]$, and secondly to normalize each original list of feature $v_{orig}[]$ as follows:

$$v_{norm}[i] = \frac{v_{orig}[i] - min[i]}{max[i] - min[i]} \quad (1)$$

Moreover, since the considered histograms present feature values that do not span over all range of admissible values, we preliminary apply an additional transformation, based on the logarithmic function, that allows us to spread the feature values more uniformly in the interval $[0, 1]$ (this is applied also to the arrays $min[]$ and $max[]$).

$$v_{log}[i] = log(1 + c * v_{orig}[i]) \quad (2)$$

The constant value c can be set to 1 or a greater value in order to increase the effect of the logarithmic function.

Finally, relatively to point (3), we tested both kinds of mentioned autoencoders: the stacked and the convolutional ones, for the first one we choose 3 fully connected layers, while for the second one we use 3 convolutional layers. We tested different values for the embedding dimension (also called *latent dimension*), and different numbers of nodes in each layer.

In the next section we present the results of the experiments that we performed to select the model $M1$ and the first results about the training of a second model $M2(o, i, p)$ estimating the selectivity (p) of range queries (o) considering the implementation (i) in Spark.

3 EXPERIMENTS

Considering the architecture of the proposed approach, we divide the experiments in two parts. First, we need to train models for obtaining a set of candidate autoencoders ($M1$) to generate embeddings. Second, we need to train models for estimating the selectivity of range queries ($M2$).

3.1 Spatial Embeddings

The training of model $M1$ requires to prepare the histograms for the datasets described in the previous section. Each histogram is a grid 128×128 containing in each cell 6 features (see Sec. 2). Given such set of input data points, we perform two sets of experiments in order to find the best architecture for $M1$.

In the first set of experiments, denoted as Exp^1_{M1}, we consider for the training *stacked autoencorders* composed of three dense layers, since we want to test whether the synthetic generation of datasets might be sufficient to produce good autoencoders also for the real datasets. Tab. 1 reports the results obtained for the two autoencoders selected by Exp^1_{M1}, which are denoted as AE_{S1} and AE_{S2}, by varying: the values of the hyperparameters (column **Hyperp.**), i.e., the number of neurons in each fully connected layer, and the latent dimension (column **LD**), i.e., the dimension of the produced spatial embedding.

For the evaluation we use the *WMAPE accuracy measures*, namely the Weighted Mean Absolute Percentage Error, since it allows us to correctly treat zeros in the set of actual and predicted values. $WMAPE$ is calculated as $(\sum |A - P| / \sum A)$ where A is the actual value, while P is the predicted value. More specifically, we compute the $WMAPE$ error on a test set covering the 20% of the data points that the model has not seen before. Notice that, the metric $WMAPE$

Table 1: Characteristics of the selected autoencoders for extracting spatial embeddings starting from a *histogram of* $128 \times 128 \times 6$. The training has been performed with 50 epoches. Models with subscript $S*$ are stacked autoencoders, while models with subscript $C*$ are convolutional autoencoders.

Experiment	Autoencoder	Latent Dimension	Hyperparameters	Train. time (sec)	LOSS	VAL LOSS	WMAPE
Exp^1_{M1}	AE_{S1}	384	1024,512	105	9.6E-04	1.5E-03	0.363
Exp^1_{M1}	AE_{S2}	1536	1024,512	104	1.1E-03	1.6E-03	0.356
Exp^2_{M1}	AE_{C1}	768	filter(128,64)	121	1.3E-03	1.3E-03	0.352
Exp^2_{M1}	AE_{C2}	3072	filter(64,32)	80	9.8E-04	9.9E-04	0.319

for fully connected models (stacked autoencoders) is very good, in the best case it is around 0.36.

In the second set of experiments, denoted as Exp^2_{M1}, we perform the training considering CNN autoencoders. As shown in Tab. 1, in this case the models produce a sightly better quality during reconstruction (i.e., $WMAPE$ is arount 0.32 in the best case).

However, since we are not specifically interested in the reconstruction capabilities of the autoencoders, but in their ability to properly distill the characteristics of a spatial datasets, we decide to not discard the stacked autoencoders entirely. In particular, given this first session of experiments, we select the 4 models in Tab. 1 for the generation of the embeddings describing the input datasets.

In the second part of experiments, we focused on the definition of specific models for estimating the selectivity for range queries.

3.2 Range query

Given the collection of considered datasets, we randomly generate 50 query windows for each of them. Such windows are located in the reference space $(0, 0, 10, 10)$ and have an area between $5.0\text{x}10^{-4}$ and $1.3\text{x}10^{-1}$. Therefore, 100,000 range queries Q_i were executed, collecting for each one the selectivity ($\sigma(Q_i)$). Starting from them and considering the 4 autoencoders in Tab. 1, we generated 4 different sets of data points with different size according to the corresponding embedding latent dimension. The size of the obtained input dataset varies from about 0.044Gb for a latent dimension of 48, to 2.31Gb for a latent dimension of 3072.

The input of M2 is composed of two parts: the embedding of the dataset (I_e) and 8 values representing the MBR of the dataset and of the query window (I_{mbr}). Given such structure, we consider two alternative approaches for the M2 network architecture: in the first one ($M2_{dnn}$) input I_e is processed by a DNN composed of three fully connected layers, then the obtained result is concatenated with input I_{mbr} and two additional fully connected layers produce the final estimate. Conversely, in the second architecture ($M2_{cnn}$) input I_e is processed by two CNN layers and the result is concatenated with I_{mbr} and given to the same final two dense layers. In both cases we tested different configuration of the hyperparameters and combinations with the 4 embeddings mentioned at the end of the previous section. The best results are obtained with the model $M2_{dnn}$ with hyperparameters (64,32,32,32,16) and embedding AE_{S1} which produces a WMAPE of 0.405, while for $M2_{cnn}$ with hyperparameters (1024,512,512,512,25) and embedding AE_{S1} we obtain a WMAPE of 0.453. Both models increases the accuracy of the baseline which has an error or 1.32. As baseline we use the theoretical formula proposed in [2] for estimating the selectivity of range query.

4 CONCLUSION AND FUTURE WORK

In this paper we propose a new approach for exploiting the machine learning and deep learning techniques in optimization of spatial queries. The key idea is to introduce the concept of spatial embedding that is generated by the first model, $M1$, of the framework and it is used by all successive models that predict a specific cost parameter of a specific implementation of one operation. As a preliminary result, we demonstrate the this idea can be applied successfully for the estimation of selectivity of range queries, obtaining an accuracy which is better than the considered baselines.

REFERENCES

[1] W. Aref and H. Samet. 1994. A Cost Model for Query Optimization Using R-Trees. In *ACMGIS*. ACM, 60–67.
[2] Alberto Belussi and Christos Faloutsos. 1998. Self-spacial Join Selectivity Estimation Using Fractal Concepts. *ACM TIS* 16, 2 (1998), 161–201.
[3] Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2018. Detecting Skewness of Big Spatial Data in SpatialHadoop *(SIGSPATIAL '18)*. 432–435.
[4] A Belussi, S Migliorini, and A Eldawy. 2020. A Cost Model for Spatial Join Operations in SpatialHadoop. *GeoInformatica* 24, 4 (2020), 1021–1059.
[5] Ahmed Eldawy et al. 2021. Beast: Scalable Exploratory Analytics on Spatio-temporal Data. In *CIKM*. ACM.
[6] A. Eldawy and M. F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *ICDE*. 1352–1363.
[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] 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*.
[9] 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).
[10] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. 2016. Autoencoding beyond Pixels Using a Learned Similarity Metric *(ICML'16)*. JMLR.org, 1558–1566.
[11] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Learning to Steer Query Optimizers. *SIGMOD*.
[12] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *aiDM@SIGMOD*. ACM, 3:1–3:4.
[13] Ryan C. Marcus et al. 2019. Neo: A Learned Query Optimizer. *PVLDB* 12, 11 (2019), 1705–1718.
[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013).
[15] Samriddhi Singla and Ahmed Eldawy. 2020. Flexible Computation of Multidimensional Histograms. In *SpatialGems* (Seattle, Washington, USA). ACM.
[16] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2020. Using Deep Learning for Big Spatial Data Partitioning. *ACM Trans. Spatial Algorithms Syst.* 7, 1 (2020), 3:1–3:37.
[17] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2021. A Learned Query Optimizer for Spatial Join *(SIGSPATIAL '21)*. 458–467.
[18] 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.
[19] Zongheng Yang et al. 2020. NeuroCard: One Cardinality Estimator for All Tables. *PVLDB* 14, 1 (2020), 61–73.
[20] 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.