Learned Index for Spatial Queries

Haixin Wang*, Xiaoyi Fu*, Jianliang Xu* and Hua Lu†
*Department of Computer Science, Hong Kong Baptist University, Hong Kong
Email: jasonxunhkbu@gmail.com, {xiaoyifu,xujl}@comp.hkbu.edu.hk
†Department of Computer Science, Aalborg University, Denmark
Email: luhua@cs.aau.dk

***Abstract*—With the pervasiveness of location-based services (LBS), spatial data processing has received considerable attention in the research of database system management. Among various spatial query techniques, index structures play a key role in data access and query processing. However, existing spatial index structures (e.g., R-tree) mainly focus on partitioning data space or data objects. In this paper, we explore the potential to construct the spatial index structure by learning the distribution of the data. We design a new data-driven spatial index structure, namely learned Z-order Model (ZM) index, which combines the Z-order space filling curve and the staged learning model. Experimental results on both real and synthetic datasets show that our learned index significantly reduces the memory cost and performs more efficiently than R-tree in most scenarios.**

I. Introduction

Location-based services are ubiquitous. Smartphones and wearable devices generate huge amounts of geospatial data that lead to a need of efficient access and process of such data [23]. In the literature, various index structures have been designed and applied in different applications [12]. For example, R-tree is an extension of B+-tree for indexing multi-dimensional data such as geographical information and multimedia objects; R+-tree allows an object to be indexed in multiple sub-trees and avoids the overlapping of internal nodes; and kd-tree recursively divides data objects in a k-dimensional space and is efficient for information search involving multidimensional search keys.

However, these conventional spatial index structures do not take advantage of common patterns existing in real-world data. For example, searching in an R-tree always needs to traverse through the tree structure to find desired data, even for uniformly distributed datasets. On the other hand, if an index is able to capture the exact data distribution, both the look-up time and the index memory size would be reduced significantly. To leverage the data pattern and distribution, Kraska et al. [7] has proposed the concept of *learned index*, which utilizes machine learning approaches to learn a model that reflects the patterns and correlations in the data, thus enabling efficient search with a succinct structure.

Yet, the study for learned index has been confined to 1D data. In this paper, we focus on multi-dimensional spatial data and investigate how to learn efficient indexes for spatial queries. Different from 1D data, which can be sorted on attribute values, multi-dimensional data cannot be easily ordered. To address this challenge, we propose a new index structure, called *learned ZM index*, to explore the spatial data patterns and efficiently process spatial queries. The learned ZM index utilizes the Z-order curve to map multidimensional data into a 1D data space, and then constructs a multi-staged model index to learn the data distribution and predict the positions of desired data objects. Moreover, we develop an algorithm for processing spatial range queries based on the predicted data positions. Our experiments, using both real and synthetic datasets, show that the learned ZM index significantly reduces the memory cost and performs much faster than R-tree in most scenarios.

The rest of this paper is organized as follows. In Section II, we survey the existing studies in spatial index structures, range query processing, and artificial neural networks. In Section III, we present the learned ZM index and the corresponding query processing algorithm. The proposed index and algorithm are experimentally evaluated in Section IV. Finally, we conclude the paper with possible directions on future work in Section V.

II. Related Work

A. Spatial Index and Range Query

Spatial index is a data structure designed for efficient data access. Most spatial indexes are either space-driven or data-driven structures [18]. Space-driven structures (e.g., fixed grid index [15]) decompose the space into cells, and map the data objects based on geometric criteria. On the contrary, data-driven structures (e.g., R-tree [4]) divide the data objects into clusters and split the space by their minimum bounding rectangles (MBRs). Besides, UB tree [1], a combination of the Z-order curve [17] and B+-tree, is a balanced tree with data objects stored by Z-order.

Range query is one of the most popular queries using spatial indexes. A range query can be processed by accessing the root of the index and recursively retrieve the children nodes intersecting the query region. Markl [13] processes a range query by retrieving all Z-regions in a UB tree that are properly intersected by the query region. A recursive decomposition algorithm was put forward in [16], which forms a minimal set of Z-regions. In [20], Skopal et al. proposed a down-right-up algorithm for the UB tree based on two types of leaf optimizations. In [3], Ahmed et al. suggested a combination of R-tree and B-tree to improve the performance of spatial queries. However, these approaches do not leverage the distribution patterns of data objects to get a denser index representation.

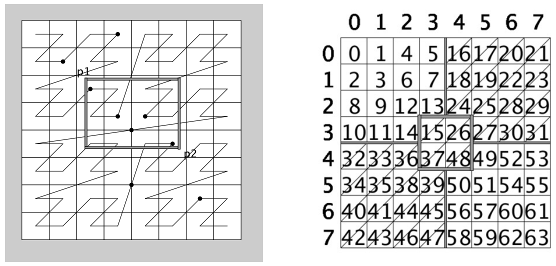
2375-0324/19/$31.00 ©2019 IEEE
DOI 10.1109/MDM.2019.00121

B. Neural Networks

Artificial neural networks (ANNs) [8] are a programming paradigm motivated by human brains, which enables computers to figure out a solution through learning from observational data. They have shown great success in many difficult tasks such as natural language processing, image recognition, and speech recognition [2, 14]. A human brain can be considered as a nonlinear and parallel information-handling system. In the system, many neurons are connected by synapses, which form a neural network. Neurons in a neural network store and process the information concurrently throughout the whole network. Inspired by the working mechanism of biological neural networks, an artificial neural network contains a number of basic processing elements called artificial neurons, or simply neurons or nodes. These elements are highly interconnected with each other, resembling the biological neurons in human brains. Analogous to neural networks, artificial neurons are connected by links that are associated with numerical weights. In an ANN, an artificial neuron gains inputs form the links, does computation on them, and then transmits the output to other neurons. By repeatedly adjusting these weights, an ANN "learns" to solve a problem.

With the increase of available training data, deep neural networks (DNNs) containing multiple layers have demonstrated significant improvement in solving complicated tasks [6]. In a DNN, neurons are put in layers that operate different functions for information transmission. In general, a DNN consists of three kinds of layers: input layer, hidden layer, and output layer. Each layer in a multilayer neural network has its own function. The input layer obtains the observable input information and transmits it to all units in the hidden layer. Then, the hidden layer detects abstract features by adjusting the weights of the neurons and sends the output to the output layer. After that, the output layer will accept the output signals from the hidden layer and form the final output pattern. DNNs have great performance in learning real-valued, discrete-valued, and vector-valued functions from given data. It has been proved in [5, 10] that an ANN with one hidden layer can approximate any bounded continuous function with arbitrarily small error, if the hidden layer has enough neurons. Besides, any function can be approximated with arbitrary error by an ANN with two hidden layers [21]. In other words, a DNN has potential to learn any function given enough hidden layers and neurons.

ANNs have received extensive research attention in recent years due to its ability to learn data distributions. Kraska et al. [7] proposed the idea of learned index, which applies ANNs to data access. They replace the traditional B-tree with a recursive model index (RMI) that consists of a number of ANN or linear models. The RMI has appreciably faster lookup speed while consuming less memory compared to a conventional B-tree in 1D data access. The innovative work of learned index inspires us to investigate spatial indexes by learned models in this paper.



(a) (b)

Fig. 1: Z-order curve and Z-address

III. LEARNED ZM INDEX

In this section, we propose a new index structure, namely *learned ZM index*, and develop spatial query techniques on the learned ZM index. First, we apply the Z-order curve and assign each point a Z-address. Then, we construct the learned ZM index that integrates Z-addresses with a multi-staged model index. Third, we present an algorithm to process spatial range queries on the learned ZM index.

A. Z-address Computation

One challenge to apply learned models to the spatial index is that there is no sequential order in multidimensional data. In a 1D data space, all data points are naturally in a linear order. Hence, it is easy to process a range query by retrieving all the data between the lower and upper bounds if the data points are sorted. For example, to find the data objects whose key values lie between α and β, we only need to find the positions of the two bounding keys and return all objects between them. However, there is no such an order for range queries in the multidimensional data space. To tackle this issue, the concepts of Z-order curve and Z-address have been proposed [17].

Specifically, the Z-order curve is a space filling curve (SFC) that maps a multidimensional vector space to a 1D space [19]. It provides a linear order for all data points in the multidimensional space. Each multidimensional vector can be converted into a unique integer called *Z-address*. Figure 1a illustrates an example of the Z-order curve. We utilize the Z-order curve for multidimensional space mapping is because it provides an important geometric property called *monotonic ordering* [9]. Before presenting the monotonic ordering property, we first introduce the concept of *dominance*.

Definition 1 (Dominance): Given two points p and p', if p is no worse than p in any dimension, then we say p dominates p'.

Property 1 (Monotonic Ordering): The data points ordered by non-descending Z-addresses are monotonic in a way that a dominating point is placed before its dominated points.

According to the monotonic ordering property, as shown in Figure 1, for a point q, the Z-address of any point dominated by q is larger than q and the Z-address of any point who dominates q is smaller than q. Hence, given a range query (specified with a rectangle with the left-top corner p_1 and the right-bottom corner p_2), retrieving all the points between p_1 and p_2 guarantees to retrieve all points in the query range.

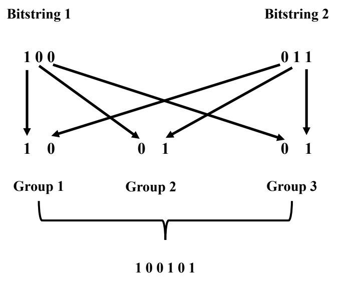


Fig. 2: Z-address computation by interleaving operation

A Z-address can be computed by interleaving the bits for each scalar value of a vector. Specifically, in an n-dimensional space, suppose that the value range of all data points is in $[0, 2^m - 1]$; then each scalar value can be represented by an m-bit bit string. For a data point, its Z-address consists of m groups and each group is represented in n bits; in total there are $m \cdot n$ bits. The i-th ($i \in [0, m]$) group in a Z-address is contributed by the i-th bit from each scalar value. Figure 1b shows the corresponding Z-addresses of the Z-order curve in Figure 1a. For example, the Z-address of $p(4, 3)$, whose binary form is (100, 011), consists of three groups 10-01-01 (37 in decimal). Figure 2 shows the process of the interleaving operation.

B. Construction of Multi-staged Model Index

The key idea of the learned ZM index is to construct a model that is able to learn the data distribution of lookup keys and effectively predict the position or existence of records. With the premise that all existing index structures can be regarded as a sort of models, machine learning provides an opportunity to approximate the data distribution with low engineering cost. For example, a typical B-tree costs $O(logn)$ lookup time and $O(n)$ index memory size, while a highly tuned learned model has the potential to achieve $O(1)$ time complexity and $O(1)$ index memory size.

As introduced in Section III-A, the data objects are sorted according to their Z-addresses, so that the data inside a query range can be retrieved efficiently. Furthermore, given a sorted dataset, a model that can effectively predict the position of a key has a good approximation of the cumulative distribution function (CDF). Given a lookup key, the position of the corresponding object can be predicted as follows:

$$p = F(key) \times N \tag{1}$$

where $F(key)$ is the learned CDF to estimate the probability that a key is no greater than the query key and N represents the total number of data objects.

Since most machine learning models are not so good in fitting the data as B-tree, the exact CDF is quite hard to learn. Fortunately, recent studies show that ANN models are considerably efficient to approximate the overall shape of a CDF [11].

Based on this, we apply a multi-staged model index (MMI), which recursively partitions the data space into sub-regions in order to reduce the estimate error [7]. As shown in Figure 3, we build a staged model consisting of several stages and each stage contains one or more smaller models. Each smaller model is an ANN or other learning models (e.g., linear model). At each stage, the model chooses another model in the next stage according to the input query key until the position is estimated at the last stage. Specifically, suppose that x is a query key and y is the corresponding position of x ($y \in [0, N)$). We refer to the model at stage 1 as f_1, and its loss function L_1 is as below:

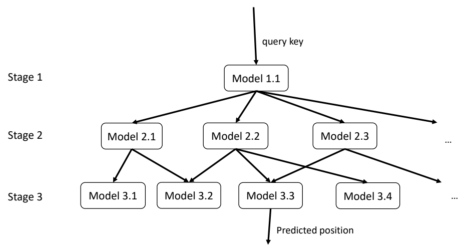


Fig. 3: Multi-staged models

$$L_1 = \sum_{(x,y)} (f_1(x) - y)^2 \tag{2}$$

Assuming that the number of models at stage i ($i > 1$) is M_i, and the jth model in stage i is donated as $f_i^{(j)}$, the loss function of model j is:

$$L_i = \sum_{(x,y)} (f_i^{(\lfloor \frac{M_i f_{i-1}(x)}{N} \rfloor)}(x) - y)^2 \tag{3}$$

The complete model can be built by recursively applying the following equation:

$$f_{i-1}(x) = f_{i-1}^{(\lfloor \frac{M_{i-1} f_{i-2}(x)}{N} \rfloor)}(x) \tag{4}$$

Each model in the staged models estimates the position of the search key with a certain error. The estimation is used to pick the model in the next stage, which is responsible for a specific subset of the whole dataset so that a prediction with a lower error could be made.

Note that the position estimated by the final stage may not be the exact position of the query key. However, it is likely to be very close to the real position of the data object. By executing the model for every key and recording the min_error (worst over-prediction) and max_error (worst under-prediction), we can provide a guarantee that one given lookup key must be in the range $[position - min\_error, position + max\_error]$ if it exists. Then, a binary search or variant can be applied to find the real position of the query key.

C. Query Processing over the learned ZM index

1) Construction of the learned ZM index: In the learned ZM index, the Z-address of each point is computed by interleaving the binary bits. All Z-addresses are indexed by an MMI. Here, the MMI index is with two stages as such a setting achieves the best performance in our experiments. The model used is a feed-forward neural network with one fully-connected hidden layer. We choose the Rectified Linear Unit (ReLU) activation function since it is able to learn various complex data distributions and has a fast convergence speed.

2) Range Query Processing: As previously mentioned, the MMI index estimates the position given a query key. The estimated position is not always the key's exact position. We now propose a search strategy to find the exact position. Different from B-tree which usually returns the pointer of a page for efficiency reasons [7], the learned model provides the prediction of the object position. As the predicted position is expected to be near to the exact position, leveraging the estimate to find the data object is expected to be faster than the conventional binary search. In this paper, we apply the Model Biased Search (MBS) [7] to find the precise position of the data object after the MMI predicts the estimated position. The MBS is a variant of binary search in which the first middle element is the estimated position.

Using the learned ZM index, we can devise effective algorithms to handle spatial queries. In this paper, we take the range query as an exemplary example as the range query is one of the most popular and practical spatial queries. It is worth noting that the learned ZM index can also support other spatial queries such as point query and kNN query.

Given a query region (donated by two spatial points p_start and p_end), a range query aims to retrieve all data objects residing in the region. The range query is formally defined as follows:

Definition 2 (Range Query): For an n-dimensional space, given a query region defined by $p\_start = (a_0, a_1, , a_{n-1})$ and $p\_end = (b_0, b_1, , b_{n-1})$ where $a_i \leq b_i$ for $0 \leq i \leq n-1$, the range query returns all spatial objects $q = (q_0, q_1, , q_{n-1})$ such that $a_i \leq q_i \leq b_i$ where $0 \leq i \leq n-1$.

Our model processes a range query based on the monotonic ordering property (Property 1). As shown in Figure 1a, given a query region donated by the left-top corner p_1 and the right-bottom corner p_2, any object q in the region satisfies the condition that $Z\_address(p_1) < Z\_address(q) < Z\_address(p_2)$. It is guaranteed to find all objects in the region if we retrieve every object whose $Z\_address$ is in the range $[Z\_address(p_1), Z\_address(p_2)]$.

Algorithm 1 presents the detailed procedure of processing a range query. The algorithm takes the points p_start and p_end that represent the query range as the input. It first computes the Z-address of p_start and p_end (lines 3-4). The function **predict** returns the estimated position of a point. Then, the algorithm finds the exact positions of the two points using the MBS search (lines 5-6). The algorithm proceeds to check each object between pos_start and pos_end, and adds the points inside the query range to the result set (lines 7-9). Note that scanning is not always efficient here. For example, in Figure 1b, even though the query region is quite small, we need to retrieve all objects in the shadowed part for the range query. However, with the optimization of partitioning the query range into multiple continuous segments, the number of data objects we need to retrieve can be significantly reduced.

Algorithm 1 Range Query Processing

```
Input: points $p\_start, p\_end$
Output: range query result $result$
 1: $data \leftarrow all\_data$
 2: $result \leftarrow \emptyset$
 3: $z\_start \leftarrow \mathbf{Z\_address}(p\_start)$    ▷ Compute the start point's Z-address
 4: $z\_end \leftarrow \mathbf{Z\_address}(p\_end)$
 5: $pos\_start = \mathbf{biased\_search}(\mathbf{predict}(z\_start))$    ▷ Point query for the start point
 6: $pos\_end = \mathbf{biased\_search}(\mathbf{predict}(z\_end))$
 7: for $s \leftarrow pos\_start$ to $pos\_end$ do
 8:     if in_region(object[$s$]) then    ▷ Check if the object is in the query region
 9:         $result \leftarrow result$+data[$s$])
10: return $result$
```

3) Theoretical Analysis: The power of the learned ZM index lies in constructing a linear order for all spatial objects and learning the distribution pattern of Z-addresses. Compared with the conventional R-tree, where search is required in non-leaf nodes to find the appropriate children, the learned ZM index has no search process in-between the stages. The result of the current stage is used to select the model in the next stage. Therefore, the search time complexity is reduced.

Furthermore, the search time complexity and index memory size are independent of the size of the dataset. Consider a dataset with N objects. R-tree requires $O(log_M N)$ search time and $O(N)$ index memory size, where M is the maximum number of data objects in a page. For the learned ZM index, N will not affect the search time and index memory size. Suppose that the learned ZM index is trained with one stage, the model in this stage is a feed-forward neural network with two hidden layers containing h and w neurons, respectively. The model scales $O(hw)$ multiplications and additions and $O(hw)$ index memory size. The models with more stages and layers will incur more engineering cost, but we can manually manage the number of stages and neurons regardless of N. In other words, if our model can learn an extremely simple but precise distribution for Z-addresses, we can achieve constant search time and index memory size. For example, if the distribution of Z-addresses is linear, our model will become a simple linear model whose search time and memory size are constant.

IV. Performance Evaluation

In this section, we experimentally evaluate the performance of the learned ZM index structure, and compare it with the standard R-tree with fan-out 100. All the algorithms are run on a computer with Intel Core i7-4770HQ 2.2GHz CPU and 16G RAM. We conduct the experiments on a synthetic dataset and a real-world dataset. The synthetic dataset (RANDOM) includes 100,000 randomly generated objects in a square Euclidean space. The real-world dataset (POST) contains the positions of 123,593 post offices in the northeast of America [22]. The data distribution of POST is shown in Figure 4. For each experiment, we measure the following metrics: the index memory size and the query processing time.

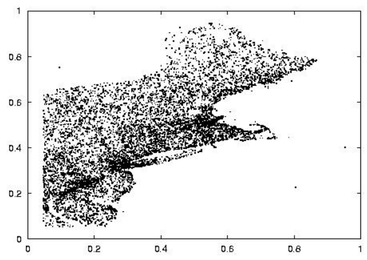


Fig. 4: The distribution of the POST dataset

TABLE I: Model parameter setting and error

Dataset	**RANDOM**		**POST**	
Model	R-tree	ZM-512	R-tree	ZM-128
Fan-out	100	N/A	100	N/A
Min-error	0	182	0	92
Max-error	0	180	0	65
Error range	0	362	0	157

We tune the learned ZM index by grid search on neural networks with one to two hidden layers. After tuning, the models with one hidden layer yield better results. Hence, we use the model with one hidden layer in the experiments. The index for the RANDOM dataset has 512 neurons (ZM-512 for short) while the model for the POST dataset has 128 neurons (ZM-128). Table I shows the configurations and errors for our index models. The min-error and max-error represent the worst over-prediction and under-prediction, respectively. The error range is the range for finding the actual position given a predicted position. As shown in Table I, our model requires less neurons to learn the data distribution of POST as well as achieving smaller errors. The distribution of the RANDOM dataset is more difficult to learn, but our model manages to keep the error range at 362, which is considered acceptable for a spatial dataset with 100,000 objects. Since an R-tree can be viewed as a model fitting the data, the error of the R-tree is 0.

A. *Index Memory Size*

Figure 5 shows the index memory size of the R-tree and our learned ZM index with different datasets. As can be seen, the learned ZM index outperforms the R-tree in both datasets. For the RANDOM dataset, our index uses 94% less memory compared with the R-tree. For the POST dataset, our model

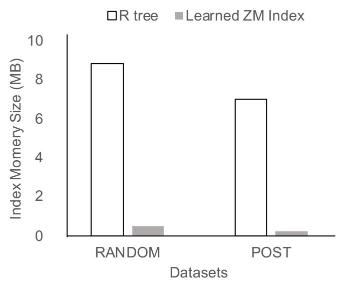


Fig. 5: Index memory size

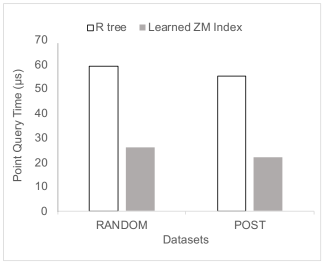
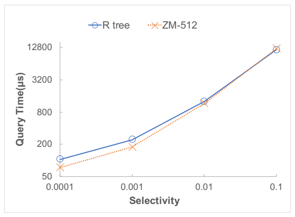
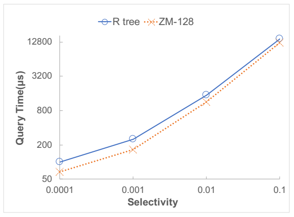


Fig. 6: Point query



(a) RANDOM



(b) POST

Fig. 7: Range Query

takes 97% less memory. This is expected, because our neural network has only a few parameters in the model. As analyzed in Section III, simple models can save the index memory size.

B. *Point Query*

We next conduct experiments for point queries on the two datasets to evaluate the efficiency of the learned ZM index. For each dataset, we randomly select 10,000 points and do point queries. The average query time is measured and shown in Figure 6. Clearly, the learned ZM index has a better search performance. The query speed of our learned ZM index model is 2.3X of the R-tree on the RANDOM dataset and 2.5X on the POST dataset. This is because our learned ZM index has a simpler structure compared with R-tree. As discussed in Section III, the predicted position by our model is directly used for data search, instead of traversing through a complex tree structure.

C. *Range Query*

Figure 7 shows the performance of range queries by varying the selectivity from 0.0001 to 0.1. Here, the selectivity represents a fraction of the number of data objects returned by a range query from the total number of data points in the dataset. It indicates how many data points from the dataset will be returned by a range query. For each selectivity setting, 50 queries are randomly generated to measure the average query time.

On the RANDOM dataset, the performance of the learned ZM index is close to the R-tree (Figure 7a). When the selectivity is low, the learned ZM index is 40% faster than the R-tree. When searching a large amount of data points (with a selectivity of 0.1), the R-tree has slightly faster query time (5% faster than the learned model). This is reasonable because the distribution of the RANDOM dataset is uncertain and there are no enough distribution features for the learned ZM index to capture. Moreover, the learned ZM index needs to decode the Z-addresses back to 2D points. When the number of data points requiring decoding becomes large, the total query cost increases.

For the POST dataset, our index achieves a better performance than the R-tree. When the query size is small, the learned ZM index achieves 50% speed-up, and when the query size is large, our index is about 17% faster than the R-tree. The reason is that with the increase of query size, most search time is consumed at scanning the objects, decoding them back to 2D data, and checking whether they are in the query range; as a result, the speed-up of the learned index becomes less obvious.

V. Conclusion

In this paper, we have shown that learned index has the potential to provide benefits on spatial data access and query processing. We have proposed a new index model, the learned ZM index, to learn the data distribution and predict the positions of data objects. We also propose efficient algorithms for processing spatial queries. Our experimental results suggest that with a well-tuned model, our learned ZM index can significantly reduce the index memory size and achieves good search performance in query time.

As for future work, several interesting directions exist. First, in addition to the ANN models, it is of interest to consider other machine learning models, such as Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Second, we plan to investigate how to support data updates by predicting the pattern of inserted data objects. Third, it is also interesting to study more spatial queries, e.g., the kNN query and join query.

Acknowledgement

This work was partially supported by HK-RGC grants 12201615, 12200817, and 12201018.

References

[1] R. Bayer. "The universal B-tree for multidimensional indexing: General concepts". In: *International Conference on Worldwide Computing and Its Applications*. Springer. 1997, pp. 198–209.

[2] S. E. Boquera, M. J. C. Bleda, J. Gorbe-Moya, and F. Zamora-Martínez. "Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models". In: *IEEE TPAMI* 33 (2011), pp. 767–779.

[3] M. A. A. Elwahab, K. M. Mahar, H. M. Abdelkader, and H. Khater. "Combining R-Tree and B-Tree to Enhance Spatial Queries Processing". In: 2013.

[4] A. Guttman. *R-trees: a dynamic index structure for spatial searching*. Vol. 14. 2. ACM, 1984.

[5] K. Hornik, M. B. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2 (1989), pp. 359–366.

[6] N. Jaitly, P. Nguyen, A. W. Senior, and V. Vanhoucke. "Application of Pretrained Deep Neural Networks to Large Vocabulary Speech Recognition". In: *INTERSPEECH*. 2012.

[7] T. Kraska, A. Beutel, E. H.-h. Chi, J. Dean, and N. Polyzotis. "The Case for Learned Index Structures". In: *SIGMOD Conference*. 2018.

[8] B. Kröse, B. Krose, P. van der Smagt, and P. Smagt. "An introduction to neural networks". In: *J Comput Sci* 48 (Jan. 1993).

[9] K. C. K. Lee, B. Zheng, H. Li, and W.-C. Lee. "Approaching the Skyline in Z Order". In: *VLDB*. 2007.

[10] G. Lewicki and G. Marino. "Approximation by Superpositions of a Sigmoidal Function". In: 2003.

[11] M. Magdon-Ismail and A. F. Atiya. "Density estimation and random variate generation using multilayer networks". In: *IEEE transactions on neural networks* 13 3 (2002), pp. 497–520.

[12] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Aristotle. "R-trees Have Grown Everywhere". In: 2003.

[13] V. Markl. "Mistral: Processing relational queries using a multidimensional access technique". In: *Ausgezeichnete Informatikdissertationen 1999*. Springer, 2000, pp. 158–168.

[14] M. A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018. URL: http://neuralnetworksanddeeplearning.com/.

[15] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. "The grid file: An adaptable, symmetric multikey file structure". In: *ACM TODS* 9.1 (1984), pp. 38–71.

[16] J. A. Orenstein and T. H. Merrett. "A Class of Data Structures for Associative Searching". In: *ACM*. PODS '84. Waterloo,Canada: ACM, 1984, pp. 181–190.

[17] F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, and R. Bayer. "Integrating the UB-tree into a database system kernel." In: *VLDB*. Vol. 2000. 2000, pp. 263–272.

[18] P. Rigaux, M. Scholl, and A. Voisard. *Spatial databases: with application to GIS*. Elsevier, 2001.

[19] H. Sagan. "Hilbert's Space-Filling Curve". In: Jan. 1994, pp. 9–30.

[20] T. Skopal, M. Krátkỳ, J. Pokornỳ, and V. Snášel. "A new range query algorithm for universal B-trees". In: *Information Systems* 31.6 (2006), pp. 489–511.

[21] UIUC and G. Cybenko. "Continuous valued neural networks with two hidden layers are sufficient." In: (1988).

[22] J. Xu, B. Zheng, W.-C. Lee, and D. L. Lee. "The D-tree: an index structure for planar point queries in location-based wireless services". In: *IEEE TKDE* 16 (2004), pp. 1526–1542.

[23] Q. Zhu, H. Hu, J. Xu, and W.-C. Lee. "Geo-social group queries with minimum acquaintance constraints". In: *The VLDB Journal* 26 (2017), pp. 709–727.