Indexing

Jianzhong Qi and Rui Zhang

Synonyms

Big spatial data access methods; Indexing big spatial data

Definitions

Consider a set of n data objects $O = \{o_1, o_2, \ldots, o_n\}$. Each object is associated with a d-dimensional vector representing its coordinate in a d-dimensional space ($d \in \mathbb{N}_+$). *Indexing* such a set of data is to organize the data in a way that provides fast access to the data, for processing spatial queries such as *point queries*, *range queries* (window queries), *kNN queries*, *spatial join queries*, etc.

Overview

The proliferation of mobile devices, ubiquitous connectivity, and *location-based services* (LBS) has accumulated a massive amount of spatial data such as user GPS coordinates, which calls for efficient indexing structures to provide fast access to such data. Typical applications of spatial data include digital mapping services and location-based social networks, where spatial queries are issued by users such as *ranking nearby restaurants by their distances to Alice* or *finding other users within 300 meters around Alice*. Spatial

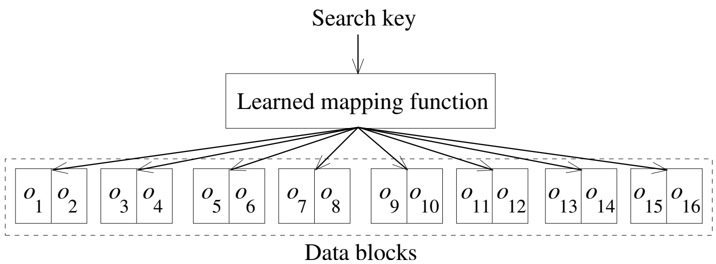
J. Qi • R. Zhang
The University of Melbourne, Melbourne, VIC, Australia e-mail: jianzhong.qi@unimelb.edu.au; e-mail: rui.zhang@unimelb.edu.au

indices have been studied extensively to support such queries. In the big data era, there may be millions of spatial queries and places of interests to be processed at the same time. This poses new challenges in both scalability and efficiency of spatial indexing techniques. Parallel spatial index structures are developed to address these challenges. Most recently, machine learning-based techniques are introduced to spatial indexing, resulting in so-called *learned spatial indices*. These techniques are summarized in the following sections.

Key Research Findings

Traditional spatial indexing techniques can be classified into *space partitioning* techniques, *data partitioning* techniques, and *mapping-based* techniques. Space partitioning techniques such as *quad trees* (Finkel and Bentley 1974) recursively partition the data space into nonoverlapping partitions where data objects in the same space partition are indexed together. Data partitioning techniques such as *R-trees* (Guttman 1984) partition a given data set directly based on the clustering relationship of the data objects. The partitions are combined recursively to form nodes of the indices. Mapping-based techniques such as *SSI* (Zhang et al. 2014) map spatial (multidimensional) data into one-dimensional values, which are then indexed by a one-dimensional index structure such as B-trees. Gaede and Günther (1998) offer a detailed survey on these traditional indices. Building upon these traditional indices, later studies design indices for various application scenarios, such as indexing temporal data (Lomet et al. 2008; Zhang and Stradling 2010), high dimensional data (Zhang et al. 2004; Jagadish et al. 2005), and moving object data (Jensen et al. 2006; Zhang et al. 2008), just to name but a few.

Spatial indices have been extended to parallel computing environments to cope with the scalability and efficiency issues. For example, the R-trees have been implemented over a shared-nothing (client-server) architecture. Koudas et al. (1996) store the inner nodes of an R-tree on a server and the leaf nodes on clients. The inner nodes on the server form a *global index*. At query processing, the server uses this index to prune the search space and locate the clients that contain the data objects being queried. Schnitzer and Leutenegger (1999) further create local R-trees on clients. This forms a two-level index structure where the global index is hosted on the server while the *local indices* are hosted on the clients. More recent works use the *MapReduce* and the *Spark* frameworks which have become standard parallel computation frameworks in the past decade. These frameworks hide the parallelism details such as synchronization and simplify parallel computation into a few generic operations (e.g., *Map* and *Reduce*). The two-level index structure is still commonly used, although some techniques only use either the global or the local indices.



Indexing, Fig. 1 Learned index

Most recently, a seminal study (Kraska et al. 2018) takes advantage of machine learning techniques and proposes *learned indices* over one-dimensional data. This technique has been extended to spatial data. Learned indices treat an index as a function $\mathcal{F}$ to map a search key to the storage address of a data object (cf. Fig. 1). Once learned, function $\mathcal{F}$ can process a point query by a function invocation with a high efficiency – constant time in ideal case. Kraska et al. (2018) learn function $\mathcal{F}$ using a *feedforward neural network*. They first sort the data points. Then, function $\mathcal{F}$ maps a search key $o_i.key$ to the *rank* $o_i.rank$ (a percentage value) of the corresponding data point o_i. The learned function $\mathcal{F}$ is essentially a *cumulative distribution function* (CDF) of the data set. The address of o_i is computed as $\mathcal{F}(o_i.key) \cdot n$.

Next, we describe a few representative spatial index structures using parallel frameworks and machine learning models, respectively.

Parallel Framework-Based Indexing

MapReduce-based indexing. *Hadoop-GIS* (Aji et al. 2013) is a Hadoop-based spatial data warehousing system, where Hadoop is an open-source MapReduce implementation. It builds a two-level (global and local) index to support parallel processing of spatial queries. The global index is pre-built by first partitioning the data space with a regular grid. Then, a grid cell is further partitioned into two equi-sized cells recursively until each cell contains no more than a predefined number of objects. Data objects in the same cell are associated with the same unique id and are stored together. Multiple cells are grouped into a block for storage to suit the block size in the *HDFS* (a distributed file system used by Hadoop). The *minimum bounding rectangles* (MBR) of the grid cells are stored in the global index. A local index is built at query time over the data objects on each machine for query processing. *SpatialHadoop* (Eldawy and Mokbel 2013) extends Hadoop to provide built-in spatial query support. It also uses a global index and a local index. *AQWA* (Aly et al. 2015) builds a global index only, which uses the k-d tree.

When a hierarchical index such as the R-tree or the k-d tree is built with MapReduce, inserting data objects into the index individually lacks efficiency. A common assumption is that the entire data set is available, and the index is *bulk-loaded* at once. Achakeev et al. (2012) bulk-load an R-tree level by level, where each level incurs a round of MapReduce computation. Qi et al. (2018; 2020) bulk-load multiple levels of an R-tree in each MapReduce round and reduce the number of rounds to $O(\log_s n)$. Here, s denotes the number of data points that can be processed by a single machine in the MapReduce cluster. Agarwal et al. (2016) bulk-load k-d trees. In each round, they use a sample data set small enough for a single machine to build a k-d tree. Data objects are assigned to the partitions of this k-d tree. For the partitions that contain too many data objects to be stored together, another round of sampling- and k-d tree-based partitioning is performed. This procedure takes $O(\texttt{poly}\log_s n)$ MapReduce rounds to bulk-load a k-d tree.

Spark-based indexing. *Spark* models a data set as a *resilient distributed dataset* (RDD) and stores it across machines in a cluster. Spatial indices on Spark focus on optimizing with the RDD storage. *GeoSpark* (Yu et al. 2015) builds a local index (e.g., a quad tree) on-the-fly for the data objects in each RDD partition, while the partitions are created by a uniform grid over the data space. *SpatialSpark* (You et al. 2015) also builds a local index for each RDD partition, which can be stored together with the data objects in the RDD partition. It supports binary space partitioning and tile partitioning in addition to uniform grid partitioning over the data space. *STARK* (Hagedorn et al. 2017) uses R-trees for the local indices while uniform grid partitioning and cost-based binary space partitioning for creating the partitions. *Simba* (Xie et al. 2016) uses global and local indices. The global index is held in-memory in the master node, while local indices are held in RDD partitions.

Machine Learning-Based Indexing

The *Z-order model* (Wang et al. 2019) extends learned indices (Kraska et al. 2018) to spatial point data. It orders data points by a *space-filling curve* (SFC), in particular, a *Z-curve* (Orenstein and Merrett 1984). An SFC imposes a grid over the data space and goes through each grid cell exactly once. This gives a curve value (a *Z-value* for Z-curves) to each cell. Data points in the same cell share the same curve value of the cell. The Z-values are used as the search key, and the index function $\mathcal{F}$ learns to predict the rank of point o_i given its Z-value. At query time, a query point is first mapped to its Z-value. This is done by interleaving the bits of its coordinates. Then, function $\mathcal{F}$ is invoked to predict the rank (address) of the query point.

Another learned index for multidimensional (spatial) data is proposed as part of a learned database system named *SageDB* (Kraska et al. 2019). To learn a multidimensional index, SageDB sorts and partitions the data points successively along a sequence of dimensions with equi-sized cells. The data points are then ordered by their cells, based on which an index function $\mathcal{F}$ is learned. To optimize query performance, SageDB also learns the dimensions used for sorting the data points as well as the partition granularity.

Nathan et al. (2019a; 2019b) learn a multidimensional index by first partitioning a d-dimensional space with a $(d-1)$-dimensional grid. They then apply the learned index technique on the d-th dimension for the data points in each grid cell. They use a *cell table* to record the coordinate range of each grid cell. Given a (range) query, this cell table returns the cells intersected by the query. The returned cells are queried further to compute the query result using their learned indices on the dimension-d coordinates.

Two other learned structures over multidimensional data have been proposed: Macke et al. (2018) learn *Bloom filters* for existence queries, while Dong et al. (2020) learn *neural locality-sensitive hashing* for kNN queries.

Examples of Application

Spatial indices are used in spatial databases to support applications in both business and daily scenarios such as targeted advertising, digital mapping, augmented reality gaming, and location-based social networking. In these applications, the locations (coordinates) of large sets of places of interests as well as users are stored in a spatial database. Spatial indices are built on such data to provide fast data access, facilitating spatial queries such as *finding customers within 100 meters of a shopping mall* (to push shopping vouchers), *ranking restaurants by their distances to Alice* (for recommendations), *finding game players in a nearby region* (for gaming interactions), and *finding users in nearby suburbs with common interests* (for friendship recommendations).

Future Directions for Research

The indexing techniques discussed above do not consider location data with frequent updates or trajectory data. Such types of data are becoming more and more common with the increasing popularity of smart phones, which enables tracking and querying the continuously changing locations of users or data objects (Li et al. 2015; Li et al. 2019; Shang et al. 2019; Xu et al. 2019). How to update the indices and to record multiple versions of the data (Lomet et al. 2008) with a high frequency is nontrivial. While the parallel frameworks scale well to massive data, the indices stored in the distributed storage may not be updated as easily as indices stored in a standalone machine. Machine learning models are also difficult to update. A periodic index rebuild (or model retraining) is an option, but this may generate inaccurate query answers between rebuilds. More efficiency update techniques are in need. Another direction to explore is to take advantage of graphics processing units (Li et al. 2018; Dong et al. 2020), which offer massively parallel processing power and are advancing rapidly as driven by the development of deep learning techniques. Further, emerging applications call for support of more query types, especially those involving multiple data sets (Zhang et al. 2012; Ward et al. 2014; Tong et al. 2018; Li et al. 2019). Particularly, prediction-based query algorithms for learned indices may not apply to such queries, and novel query algorithms are needed. For more challenges and opportunities in machine learning techniques for big spatial data, interested readers are referred to two surveys (Karpatne et al. 2019; Sabek and Mokbel 2019).

Cross-References

Query Processing: Computational Geometry
Query Processing: Joins
Query Processing – kNN

References

Achakeev D, Seidemann M, Schmidt M, Seeger B (2012) Sort-based parallel loading of r-trees. In: Proceedings of the 1st ACM SIGSPATIAL international workshop on analytics for big geospatial data, pp 62–70

Agarwal PK, Fox K, Munagala K, Nath A (2016) Parallel algorithms for constructing range and nearest-neighbor searching data structures. In: Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems (PODS), pp 429–440

Aji A, Wang F, Vo H, Lee R, Liu Q, Zhang X, Saltz J (2013) Hadoop gis: a high performance spatial data warehousing system over mapreduce. Proc VLDB Endow 6(11):1009–1020

Aly AM, Mahmood AR, Hassan MS, Aref WG, Ouzzani M, Elmeleegy H, Qadah T (2015) Aqwa: adaptive query workload aware partitioning of big spatial data. Proc VLDB Endow 8(13):2062–2073

Dong K, Zhang B, Shen Y, Zhu Y, Yu J (2020) GAT: a unified gpu-accelerated framework for processing batch trajectory queries. IEEE Trans. Knowl. Data Eng. 32(1):92–107

Dong Y, Indyk P, Razenshteyn I, Wagner T (2020) Learning space partitions for nearest neighbor search. In: The 8th international conference on learning representations (ICLR)

Eldawy A, Mokbel MF (2013) A demonstration of spatialhadoop: an efficient mapreduce framework for spatial data. Proc VLDB Endow 6(12):1230–1233

Finkel RA, Bentley JL (1974) Quad trees a data structure for retrieval on composite keys. Acta Informatica 4(1):1–9

Gaede V, Günther O (1998) Multidimensional access methods. ACM Comput Surv 30(2):170–231

Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD), pp 47–57

Hagedorn S, Götze P, Sattler K (2017) Big spatial data processing frameworks: feature and performance evaluation. In: Proceedings of the 20th international conference on extending database technology (EDBT), pp 490–493

Jagadish HV, Ooi BC, Tan KL, Yu C, Zhang R (2005) Idistance: an adaptive b+-tree based indexing method for nearest neighbor search. ACM Trans Database Syst 30(2):364–397

Jensen CS, Lin D, Ooi BC, Zhang R (2006) Effective density queries on continuously moving objects. In: Proceedings of the 22nd IEEE international conference on data engineering (ICDE), pp 71–82

Karpatne A, Ebert-Uphoff I, Ravela S, Babaie HA, Kumar V (2019): Machine learning for the geosciences: Challenges and opportunities. IEEE Trans. Knowl. Data Eng. 31(8): 1544–1554

Koudas N, Faloutsos C, Kamel I (1996) Declustering spatial databases on a multi-computer architecture. In: Proceedings of the 5th international conference on extending database technology (EDBT), pp 592–614

Kraska T, Alizadeh M, Beutel A, Chi EH, Kristo A, Leclerc G, Madden S, Mao H, Nathan V (2019) SageDB: a learned database system. In: The 9th biennial conference on innovative data systems research (CIDR)

Kraska T, Beutel A, Chi EH, Dean J, Polyzotis N (2018) The case for learned index structures. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD), pp 489 –504

Li C, Gu Y, Qi J, He J, Deng Q, Yu G (2018) A gpu accelerated update efficient index for knn queries in road networks. In: Proceedings of the 34th international conference on data engineering (ICDE), pp 881–892

Li C, Gu Y, Qi J, Zhang R, Yu G (2015) A safe region based approach to moving knn queries in obstructed space. Knowl Inf Syst, 45(2):417–451

Li C, Gu Y, Qi J, Zhang R, Yu G (2019) Moving knn query processing in metric space based on influential sets. Inf Syst, 83:126–144

Li Y, Eldawy A, Xue J, Knorozova N, Mokbel MF, Janardan R (2019) Scalable computational geometry in MapReduce. VLDB J, 28:523–548

Lomet D, Hong M, Nehme R, Zhang R (2008) Transaction time indexing with version compression. Proc VLDB Endow 1(1):870–881

Macke S, Beutel A, Kraska T, Sathiamoorthy M, Cheng DZ, Chi EH. (2018) Lifting the curse of multidimensional data with learned existence indexes, In: NeurIPS workshop on machine learning for systems

Nathan V, Ding J, Alizadeh M, Kraska T (2019) Learning multi-dimensional indexes. In: NeurIPS workshop on machine learning for systems

Nathan V, Ding J, Alizadeh M, Kraska T (2019) Learning multi-dimensional indexes. CoRR,abs/1912.01668

Orenstein JA, Merrett TH (1984) A class of data structures for associative searching. In: Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on principles of database systems (PODS), pp 181–190

Qi J, Tao Y, Chang Y, Zhang R (2018) Theoretically optimal and empirically efficient r-trees with strong parallelizability. Proc VLDB Endow 11(5): 621–634

Qi J, Tao Y, Chang Y, Zhang R (2020) Packing r-trees with space-filling curves: theoretical optimality, empirical efficiency, and bulk-loading parallelizability. ACM Trans Database Syst

Sabek I, Mokbel MF (2019) Machine learning meets big spatial data. Proc. VLDB Endow 12(12):1982–1985

Schnitzer B, Leutenegger ST (1999) Master-client r-trees: a new parallel r-tree architecture. In: Proceedings of the 11th international conference on scientific and statistical database management (SSDBM), pp 68–77

Shang S, Chen L, Zheng K, Jensen CS, Wei Z, Kalnis P (2019) Parallel trajectory-to-location join. IEEE Trans. Knowl. Data Eng. 31(6):1194–1207

Tong Y, Wang L, Zhou Z, Chen L, Du B, Ye J (2018) Dynamic pricing in spatial crowdsourcing: A matching-based approach. In: Proceedings of the 2018 international conference on management of data (SIGMOD), pp 773–788

Wang H, Fu X, Xu J, Lu H (2019) Learned index for spatial queries. In: Proceedings of the 20th IEEE international conference on mobile data management (MDM), pp 569–574

Ward PGD, He Z, Zhang R, Qi J (2014) Real-time continuous intersection joins over large sets of moving objects using graphic processing units. VLDB J 23(6):965–985

Xie D, Li F, Yao B, Li G, Zhou L, Guo M (2016) Simba: efficient in-memory spatial analytics. In: Proceedings of the 2016 SIGMOD international conference on management of data (SIGMOD), pp 1071–1085

Xu H, Gu Y, Sun Y, Qi J, Yu G, Zhang R (2019) Efficient processing of moving collective spatial keyword queries. VLDB J

You S, Zhang J, Gruenwald L (2015) Large-scale spatial join query processing in cloud. In: Proceedings of the 31st IEEE international conference on data engineering workshops, pp 34–41

Yu J, Wu J, Sarwat M (2015) Geospark: a cluster computing framework for processing large-scale spatial data. In: Proceedings of the 23rd SIGSPA-

TIAL international conference on advances in geographic information systems (SIGSPATIAL), pp 70:1–70:4

Zhang R, Lin D, Ramamohanarao K, Bertino E (2008) Continuous intersection joins over moving objects. In: Proceedings of the 24th IEEE international conference on data engineering (ICDE), pp 863–872

Zhang R, Ooi BC, Tan KL (2004) Making the pyramid technique robust to query types and workloads. In: Proceedings of the 20th IEEE international conference on data engineering (ICDE), pp 313–324

Zhang R, Qi J, Lin D, Wang W, Wong RC (2012) A highly optimized algorithm for continuous intersection join queries over moving objects. VLDB J 21(4):561–586

Zhang R, Qi J, Stradling M, Huang J (2014) Towards a painless index for spatial objects. ACM Trans Database Syst 39(3):19:1–19:42

Zhang R, Stradling M (2010) The hv-tree: a memory hierarchy aware version index. Proc VLDB Endow, 3(1-2):397–408