Query Processing – kNN (2020 revision)

Jianzhong Qi and Rui Zhang

Synonyms

K Nearest Neighbor Query; kNN Query

Definitions

Consider a set of n data objects $O = \{o_1, o_2, \ldots, o_n\}$ and a query object (user) q. Each object, including the query object, is associated with a d-dimensional vector representing its coordinate in a d-dimensional space ($d \in \mathbb{N}_+$). Given a query parameter k ($k \in \mathbb{N}_+$), the k *nearest neighbor* (kNN) query computes a size-k subset $S \subseteq O$ that contains the data objects that are the nearest to q:

$$\forall o_i \in S, o_j \in O \backslash S : \text{dist}(q, o_i) \leq \text{dist}(q, o_j)$$

Here, dist$(\cdot)$ is a function that returns the distance between two objects. A variety of distance functions have been considered in the literature, such as *Euclidean (L_2) distance*, *Manhattan (L_1) distance*, *Chebyshev (L_∞) distance*, spatial network distance, and *general metric distance*.

kNN query processing in the spatial data management context has focused on two- or three-dimensional Euclidean space and spatial network space.

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

Overview

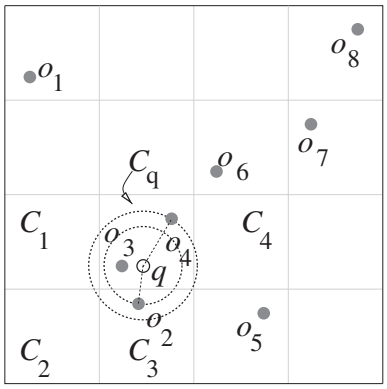
The proliferation of mobile devices, ubiquitous connectivity, and *location-based services* (LBS) has accumulated a massive amount of spatial data such as mapping data and user trajectories. Popular LBS such as Google Maps are serving millions of users who issue queries such as *finding Alice her nearest petrol stations*, which is a typical example of kNN queries. Most kNN algorithms use *spatial indices* over the data objects to help prune the unpromising search space and obtain high query efficiency. When the amount of data becomes too large to be indexed or queried on a single machine, parallelism is exploited to overcome such limitations. This raises new challenges of kNN queries in parallel index management and query processing. Most recently, machine learning-based techniques are introduced to database query processing including kNN query processing. These query processing techniques are summarized in the following sections.

Key Research Findings

We start with a brief review of two classic kNN algorithms and then focus on the discussion of recent developments on kNN algorithms that use parallelism and machine learning to manage the growing amount of data.

Traditional kNN Algorithms

The two arguably most popular kNN query algorithms are the *best-first* (Hjaltason and Samet 1995) and the *depth-first* (Roussopoulos et al. 1995) algorithms. These two algorithms rely on an tree index, e.g., an R-tree (Guttman 1984), over the data set O. They traverse the tree index starting from the tree root to find data objects nearest to q. During the traversal, the distance to the kth NN among the data objects already visited is used to facilitate the pruning of unpromising tree branches that do not index nearer data objects. When a tree node N is visited, the best-first algorithm inserts all child nodes of N into a queue Q prioritized by an estimated distance between each child node and q. The next node to be visited is the one in Q with the smallest estimated distance to q. The depth-first algorithm simply visits the first child of each node before visiting the sibling nodes. While these two algorithms have shown high empirical query efficiency due to the pruning power of the tree indices, they are difficult to be parallelized because of the backtracking procedure in traversal.



Query Processing – *k*NN (2020 revision), Fig. 1 Grid-based kNN computation

Parallel *k*NN Algorithms

Parallel kNN query algorithms that suit popular parallel computation models such as *MapReduce* (Dean and Ghemawat 2008) have been developed to handle data beyond the processing capability of individual machines. Such algorithms partition the data space with a grid (a.k.a. *tiling*, cf. Fig. 1) so that data objects in different grid cells can be examined in parallel. A two-stage procedure is used in query processing. In the first stage, the cell C_q that contains the query object q is identified, and kNN is computed over the data objects in C_q (e.g., the 2NNs in Fig. 1 are o_3 and o_4). In the second stage, the cells adjacent to C_q are examined to find data objects that may be nearer to q than the kth NN already found (e.g., cells C_1, C_2, C_3, and C_4, which are within the circle defined by the second NN o_4; o_2 is found to be the new second NN). This step may be repeated over more cells adjacent to the cells examined, until the nearest data object in these adjacent cells is farther away than the kth NN already found.

Parallel Framework-Based Algorithms

Many of the modern parallel kNN algorithms are based on *Hadoop* and *Spark* which are popular parallel frameworks.

Hadoop is an open-source implementation of the MapReduce computation model. The MapReduce model processes a task (a MapReduce job) in two phases: *Map* and *Reduce*. The Map phase assigns data to different machines, while the Reduce phase runs computation on each machine. The data storage component of Hadoop, *Hadoop Distributed File System* (HDFS), is optimized for data blocks (e.g., 64 MB per block). Hadoop-based kNN algorithms need to design indices following this block size. Aji et al. (2012) propose a kNN

algorithm based on MapReduce with an assumption that the entire data set can be replicated to each machine for querying. They process multiple kNN queries in a single MapReduce job. In the Map phase, the query objects are partitioned with a grid and assigned to different machines according to their grid cell id. In the Reduce phase, the data set is replicated to each machine, upon which a local index (e.g., an R-tree) is built for query processing on each machine independently. *SpatialHadoop* (Eldawy and Mokbel 2013) employs a two-level structure that consists of a *global index* and a *local index*. The global index is built by partitioning the data space (e.g., with a grid or an R-tree) such that every partition can fit in a block in HDFS. *Minimum bounding rectangles* (MBR) of the partitions are stored in the global index which is kept in the main memory of the master node of the cluster. A local index is built on every partition and kept in HDFS. kNN queries are processed with the two-stage search expansion procedure as described earlier. The partition containing the query object is identified using the global index, where a kNN query is computed using the local index. *AQWA* (Aly et al. 2015) uses k-d trees (Bentley 1975) to build the global index and create data partitions. It stores data object counts of each partition to streamline query processing.

HBase is an open-source *key-value* store (*NoSQL* database) built upon HDFS. It hides the underlying block storage from users, and kNN algorithms based on it do not have to consider the block storage. *MD-HBase* (Nishimura et al. 2011) stores the *Z-value* (Orenstein and Merrett 1984) of a data object together with the data object as a key-value pair in HBase. The key-value pairs are partitioned by the key range, and every partition is stored as a *region*. Data in HBase is indexed with a B^+-tree-like structure, where the key range of a region is stored in an index layer called the *META* layer. MD-HBase uses the longest common prefix of the keys (in the binary form) in a region as the index key of the region. kNN queries are processed following a search expansion procedure. At start, the region with an index key that shares the longest common prefix with the Z-value of the query object q is identified, and an initial kNN answer is computed from the region. The kth NN found defines a distance from q for search expansion. *KR^+index* (Hsu et al. 2012) follows a similar query procedure but uses an *R^+-tree* (Sellis et al. 1987) to group the data objects and *Hilbert-values* (Lawder and King 2001) of tree leaf nodes as index keys. *HGrid* (Han and Stroulia 2013) constructs index keys based on a two-level grid structure. The first level uses a grid where the cells are indexed by the Z-values. The second level further partitions each first-level grid cell with a grid where the cells are indexed by their row and column numbers. The key value of a data object is formed by concatenating the index keys of the first- and second-level cells enclosing this object. The HGrid kNN algorithm starts by querying a region that may contain sufficient objects to form a kNN answer based on the data density. Then, search expansion is done to refine the answer. *COWI* and *CONI* (Cahsai et al. 2017) build a quad-tree (Finkel and Bentley 1974) over the data set where the leaf node numbers are used as the index keys of the data objects.

Spark provides an in-memory parallel framework to handle massive data. A set of data is modeled as a *resilient distributed dataset* (RDD). RDDs are stored in partitions across machines in the cluster. *GeoSpark* (Yu et al. 2015) encapsulates RDDs with an abstraction named the *spatial resilient distributed dataset* (SRDD) to store spatial data. An SRDD is partitioned using a uniform grid. Local indices such as quad-trees may be created for each partition. kNN query processing on GeoSpark is done by the grid-based search expansion strategy as discussed earlier. *Simba* (Xie et al. 2016) uses both global and local indices. It creates a new RDD abstraction called the *IndexRDD* to store a local index (e.g., an R-tree) together with the data objects in each partition, while the partitions are created by the *STR* algorithm (Leutenegger et al. 1997). The global index (e.g., an R-tree) is created over the partitions and held in-memory in the master node for fast query processing. KNN queries on Simba also follow the search expansion strategy but fetch multiple partitions at start instead of only the partition containing the query object.

Multi-Core-Based Algorithms

Multi-core processors and *graphics processing units* (GPU) have also been used to accelerate kNN queries. For example, Chen et al. (2017) speed up kNN computation for a set of queries using a GPU. Instead of simply processing one query per thread (core), they form clusters for both the query points and the data points. KNN computation of queries in the same cluster is then shared. Šidlauskas et al. (2012) consider monitoring query results over objects with location updates (e.g., moving objects). They propose a main memory index named *PGrid* that creates a grid partitioning over the data objects and exploits multi-core processors to improve the data update efficiency (and hence query result update efficiency). Li et al. (2018) also use a grid index to compute kNN queries over moving objects. They use GPUs instead of multi-core processors, and their key idea is to delay object updates until the objects are queried, so as to improve the update efficiency. Luo et al. (2019) propose the *MPR* framework to convert single-threaded kNN algorithms into algorithms that handle multiple queries and updates in parallel on multi-core machines. They use multi-layer (query and update) workload partitioning and data replication to fully exploit the power of parallelism.

Machine Learning-Based kNN Algorithms

Most recently, a seminal study (Kraska et al. 2018) takes advantage of machine learning techniques and proposes *learned indices* for query processing. The basic idea of learned indices is to treat an index as a function $\mathcal{F}$ to map a search key to the query answer. Such a function can be learned via a neural

network due to its capability to learn non-linear functions. For kNN queries, Dong et al. (2020) propose to learn a *neural locality-sensitive hashing*. They construct a *kNN graph* for a given set of data objects by connecting every object with its kNN objects. This graph is partitioned using a balanced graph partitioning algorithm into m partitions, where m is an algorithm parameter. They train a *feedforward neural network* to predict the partition id given a data object. Given a query object, the trained network predicts the partition of the object, from which the kNN result is computed. A few other recent kNN algorithms (e.g., Malkov and Yashunin 2018; Li et al. 2019b) are also based on kNN graphs, especially in the high-dimensional or metric spaces. For query processing, a walk is run on the kNN graphs, where a variety of walk strategies have been proposed aiming to reach the query result faster.

Examples of Application

KNN queries have a large variety of applications in different fields such as spatio-temporal databases, location-based services, and data mining, just to name but a few. For spatial data, a typical application is in digital mapping, e.g., Google Maps, where kNN queries may be used to find nearest *points-of-interest* (POI) for query users. A closely related application is in ridesharing, where kNN queries may be used to find nearest cars for car riders. Another application is in augmented reality gaming, e.g., Pokémon GO, where kNN queries may be used to find nearest gaming objects for game players.

Future Directions for Research

KNN queries attract extensive research interests due to their wide range of applications. Studies emerge continuously which consider various application scenarios of kNN queries, such as kNN queries where the query object, the data objects, or both may be moving (Nutanong et al. 2008; Zhang et al. 2010; Li et al. 2016; Li et al. 2019a), keyword-based kNN queries (Chen et al. 2019; Xu et al. 2019), and trajectory-based kNN queries (Shang et al. 2018; Ali et al. 2019). Traditional kNN algorithms may not handle these novel query scenarios. For example, moving kNN queries require constant updates to the query results, while trajectory-based kNN queries may have a high cost on object (i.e., trajectory) distance computation. Further investigation in these queries is needed.

KNN queries in higher-dimensional spaces (Jagadish et al. 2005; Yu et al. 2010) is another challenge. As a major application domain, data mining requires kNN computation over feature spaces that may have thousands of dimensions. Approximate kNN algorithms including hashing-based techniques

and kNN graph-based techniques are being actively explored to cope with the high processing costs in high-dimensional spaces.

Cross-References

Indexing
Query Processing: Computational Geometry
Query Processing: Joins

References

Aji A, Wang F, Saltz JH (2012) Towards building a high performance spatial query system for large scale medical imaging data. In: Proceedings of the 20th international conference on advances in geographic information systems (SIGSPATIAL), pp 309–318

Ali ME, Eusuf SS, Abdullah K, Choudhury FM, Culpepper JS, Sellis T (2019) The maximum trajectory coverage query in spatial databases. Proc VLDB Endow, 12(3):197–209

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

Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517

Cahsai A, Ntarmos N, Anagnostopoulos C, Triantafillou P (2017) Scaling k-nearest neighbours queries (the right way). In: Proceedings of the 37th IEEE international conference on distributed computing systems (ICDCS), pp 1419–1430

Chen G, Ding Y, Shen X (2017) Sweet knn: An efficient knn on gpu through reconciliation between redundancy removal and regularity. In: Proceedings of the 33rd IEEE international conference on data engineering (ICDE), pp 621–632

Chen L, Shang S, Zheng K, Kalnis P (2019) Cluster-based subscription matching for geo-textual data streams. In: Proceedings of the 35th IEEE international conference on data engineering (ICDE), pp 890–901

Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

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

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

Han D, Stroulia E (2013) Hgrid: a data model for large geospatial data sets in hbase. In: Proceedings of the 6th IEEE international conference on cloud computing (CLOUD), pp 910–917

Hjaltason GR, Samet H (1995) Ranking in spatial databases. In: Proceedings of the 4th international symposium on advances in spatial databases (SSD), pp 83–95

Hsu YT, Pan YC, Wei LY, Peng WC, Lee WC (2012) Key formulation schemes for spatial index in cloud data managements. In: Proceedings of the 13th IEEE international conference on mobile data management (MDM), pp 21–26

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

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

Lawder JK, King PJH (2001) Querying multi-dimensional data indexed using the hilbert space-filling curve. SIGMOD Rec 30(1):19–24

Leutenegger ST, Lopez MA, Edgington J (1997) Str: a simple and efficient algorithm for r-tree packing. In: Proceedings of the 13th IEEE international conference on data engineering (ICDE), pp 497–506

Li C, Gu Y, Qi J, Yu G, Zhang R, Deng Q (2016) INSQ: an influential neighbor set based moving kNN query processing system. In: Proceedings of the 32nd IEEE international conference on data engineering (ICDE), pp 1338–1341

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 IEEE international conference on data engineering (ICDE), pp 881–892

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 W, Zhang Y, Sun Y, Wang W, Li M, Zhang W, Lin X (2019) Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement. IEEE Transactions on Knowledge and Data Engineering

Luo S, Kao B, Wu X, Cheng R (2019) MPR – a partitioning-replication framework for multi-processing knn search on road networks. In: Proceedings of the 35th IEEE international conference on data engineering (ICDE), pp 1310–1321

Malkov YA, Yashunin DA (2018) Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(4):824–836

Nishimura S, Das S, Agrawal D, Abbadi AE (2011) Md-hbase: a scalable multi-dimensional data infrastructure for location aware services. In: Proceedings of the 12th IEEE international conference on mobile data management (MDM), pp 7–16

Nutanong S, Zhang R, Tanin E, Kulik L (2008) The v*-diagram: a query-dependent approach to moving kNN queries. Proc VLDB Endow 1(1): 1095–1106

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

Roussopoulos N, Kelley S, Vincent F (1995) Nearest neighbor queries. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD), pp 71–79

Shang S, Chen L, Wei Z, Jensen CS, Zheng K, Kalnis P (2018) Parallel trajectory similarity joins in spatial networks. The VLDB Journal, 27(3):395–420

Sellis TK, Roussopoulos N, Faloutsos C (1987) The r+-tree: a dynamic index for multi-dimensional objects. In: Proceedings of the 13th international conference on very large data bases (VLDB), pp 507–518

Šidlauskas S, Šaltenis S, Jensen CS (2012) Parallel main-memory indexing for moving-object query and update workloads. In: Proceedings of the SIGMOD international conference on management of data (SIGMOD), pp 37–48.

Xie D, Li F, Yao B, Li G, Zhou L, Guo M (2016) Simba: efficient in-memory spatial analytics. In: Proceedings of the 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. The VLDB Journal

Yu C, Zhang R, Huang Y, Xiong H (2010) High-dimensional knn joins with incremental updates. Geoinformatica, 14(1):55

Yu J, Wu J, Sarwat M (2015) Geospark: a cluster computing framework for processing large-scale spatial data. In: Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems (SIGSPATIAL), pp 70: 1–70:4

Zhang R, Jagadish HV, Dai BT, Ramamohanarao K (2010) Optimized algorithms for predictive range and knn queries on moving objects. Information Systems, 35(8):911–32