International Journal of Decision Support System Technology
Volume 16 • Issue 1

A Novel Query Method for Spatial Database Based on Improved K-Nearest Neighbor Algorithm

Huili Xia, College of Computer and Artificial Intelligence, Zhengzhou University of Economics and Business, China*
Feng Xue, College of Computer and Artificial Intelligence, Zhengzhou University of Economics and Business, China

ABSTRACT

Spatial database is a spatial information database and is the core component of geographic information systems (GIS). Aiming at the problem that time complexity of k-nearest neighbor (kNN) querying algorithms are proportionate to scale of training samples, an efficient query method for spatial database based on the Spark framework and the reversed k-nearest neighbor (RkNN) is proposed. Firstly, based on the Spark framework, a two-layer indexing structure based on grid and Voronoi diagram is constructed, and an efficient filtering and a refining processing algorithm are proposed. Secondly, the filtering step of proposed algorithm is used to obtain the candidates, and the refining step is used to remove the candidates. Finally, the candidate sets from different regions are merged to get the final result. Results of experiments on real-world datasets validate that the proposed method has better query performance and better stability and significantly improves the processing speed.

KEYWORDS

Big Data, Geographic Information System, Parallel Processing, Reverse K-Nearest Neighbor, Spark, Spatial Database

INTRODUCTION

A spatial database is a database system that describes, stores, and processes spatial data and the associated attribute data in which a relational database management system (RDBMS) is used to regulate spatial data, mainly solving the data interface problem between the spatial data stored in the relational database and the application program—that is, the spatial database engine (SDE) (Sveen, 2019; Baharin, & Akunne, 2020). More precisely, spatial database technology is used to address the accessing issue of the geometric attributes of the spatial data objects in the relational database (Breunig et al., 2020; Zhang et al., 2022). In addition, the spatial database can also effectively handle the integrating, querying, and managing of complex spatial information for huge volumes of data. Compared with traditional databases, spatial databases have wider

DOI: 10.4018/IJDSST.332773 *Corresponding Author

application prospects and are the core components of a geographic information system (GIS) (Choi et al., 2020; Xia et al., 2022).

With the continuous development of location-based services (LBS), spatial databases are of great significance in the fields of traffic network systems, GIS, and various decision support systems (Pant et al., 2018). The query technology of spatial data is the utmost important operation for a spatial database. Spatial data query technology refers to the process of searching for and returning of spatial objects that meet certain query constraints in a spatial database. Commonly used spatial database query operations include Exact Match Query (Jiang et al., 2019), Point Query (Wang et al., 2019), Region Query (Wan et al., 2019), and Nearest Neighbor Query (Chen et al., 2019).

The problem of nearest neighbor query is a hotspot in today's database technology, and the query results are usually spatial data or datasets that meet the query requirements. Nearest neighbor query is the basis of the spatial database query field. Because of the continuous change of query requirements, many variants have been produced, such as k-nearest neighbor (kNN), directional nearest, clustered nearest, group nearest, constrained nearest, and privacy-preserving nearest neighbors (Wang et al., 2019). The kNN and reverse nearest neighbor (RkNN) algorithms are considered to be the most fundamental and widely used query types. The kNN algorithm can be provided as an LBS service alone, such as finding the nearest restaurant or hotel, or it can be used as a basic technology to provide support for other queries, such as RkNN and search window query (Xu et al., 2021). RkNN has a good application when it comes to the "influence range" and "influence degree" of an object on other objects, such as the location problem when building new infrastructure, such as shopping malls and hospitals, and the current popular concept of targeted advertising.

Most of the current parallel RkNN query algorithms are on the basis of the Map-Reduce framework. Ji et al. (2015) introduced a distributed RkNN query processing algorithm on the basis of the inverted grid index. García et al. (2019) conducted distributed RkNN query research on Spatial Hadoop, the spatial expansion framework of Hadoop, and proposed an RkNN query algorithm MRSLICE based on Spatial Hadoop. MapReduce framework along with open-sourced applications in Hadoop are breakthroughs in the efficient processing of large-scale datasets in which data is processed through a distributed system, and program execution will not be affected by node failures. However, researchers have found in experiments that the performance of multicore devices is limited by the single node implementation, which can easily lead to too tight coupling and low scalability. The limitation of MapReduce in the Hadoop platform will cause the start-up overhead of each round of operation in the iterative calculation process to be too large, and the overhead of disk I/O will also reduce the execution efficiency (Grolinger et al., 2014). In contrast, the Spark framework overcomes the aforementioned problems, and it accelerates the processing of distributed computing through memory computing strategies. This framework allows users to store data in memory for repeated queries, making it more practical for online, iterative, and dataflow algorithms (Meng et al., 2016).

In recent years, scholars have proposed frameworks such as SpatialSpark (You et al., 2015) and GeoSpark (Yu et al., 2015). These frameworks can realize Spark-based distributed spatial range query, kNN query, and spatial join query, and it is verified by experiments that Spark-based query processing is better than the MapReduce framework. García et al. (2017) presented a query algorithm on the basis of Spatial Hadoop (RkNN-SH) and implemented it on actual datasets. In addition to the above typical spatial queries, scholars have expanded the research on variant queries based on the Spark framework, including distance join queries (García-García et al., 2020), spatio-temporal join queries (Li et al., 2021), and top-k spatial join queries (Qiao et al., 2020).

The above studies have demonstrated the superiority of the Spark framework in processing parallel spatial queries. However, more research on RkNN query based on Spark framework needs to be done. To this end, we propose an efficient query method for spatial databases based on the improved RkNN algorithm. This method includes the following innovations:

- Based on the Spark framework, the parallel RkNN query is analyzed. By leveraging the excellent features of the Voronoi diagram (VD) in terms of spatial proximity, a parallel indexing structure based on the VD is extended on the Spark framework.
- On the double-layer index structure, filtering and refining algorithms for Spark-based RkNN query is given to improve querying efficiency and accuracy in spatial database.

In this paper, we present the background knowledge and explain the proposed query method for a spatial database based on Spark and the improved RkNN algorithm. We then analyze the performance of the proposed method through a large number of experiments and compare it with other advanced algorithms. Finally, we summarize our study and briefly state the future research possibilities.

PRELIMINARIES

Spark Framework

Spark is a memory-based computing framework proposed by the AMP Lab from the University of California, Berkeley, built using the Scala language (Zaharia et al., 2012). The Spark framework provides two main architectures for parallel computing: Resilient Distributed Datasets (RDDs) creation and parallel computing operations on datasets. Among them, RDD is the core and foundation of the Spark framework, and it provides a powerful distributed memory parallel computing engine that supports fast iterative computing.

An RDD is a distributed memory abstraction that is partitioned on multiple nodes in the cluster and represents a collection of read-only partition records. Control of joint partitioning of different RDDs on multiple machines enables data shuffling between machines to be reduced. In the Spark framework, all RDDs can be established only by performing the following deterministic operations in stable physical storage or on existing RDDs:

- RDDs are obtained in a file-sharing system, such as Hadoop Distributed File System (HDFS).
- RDDs are created by the Scala collection used in the driver for paralleled computing (for example, an array object built in the driver).
- RDDs transform an existing RDD.
- Perform persist operations to change the existing RDD.

There are two kinds of paralleled operations, transform and action. The transform operation generates a new RDD from an existing RDD, and the action operation performs calculations on the RDD, return a normal type value result, or output the data in the RDD to the storage system (Ventocilla, 2019). The user's control over an RDD can also be manipulated from two aspects: caching and partitioning. Caching the RDD between multiple concurrent operations accelerates the later reuse of the RDD. An RDD also allows users to specify the partition order according to the key values and divide the data into various partitions.

Figure 1a shows the basic architecture of Spark. The computing framework includes the MapReduce framework, which has made some architectural improvements on the basis of Hadoop. The difference between Spark and Hadoop is that the intermediate output results of Spark can be stored in memory so that there is no need to read and write HDFS again. Therefore, using the Spark parallel processing framework can greatly improve the efficiency of the iterative computing process. Based on the dependency between RDDs, the fault tolerance of the entire distributed computing is guaranteed, and the application applies for the required resources, such as CPU and memory resources through Spark's resource manager. The worker nodes start the corresponding processes and wait for the master node to assign tasks, and the nodes on the cluster cooperate to complete the tasks and return the results to the application, as shown in Figure 1b.

RkNN Algorithm

The nearest neighbor (NN) query is explained as follows (Dhanabal & Chandramathi, 2011): Assume we have a set S, q is a querying point, the purpose is to search for object p within S, which is closest to q, then the nearest neighbor $NN(q)$ of q is p. This formula is formally defined in equation (1):

$$NN(q) = \{p \in S \mid \forall r \in S, Dist(q,p) \leq Dist(q,r)\} \tag{1}$$

In this equation, $Dis(q,p)$ represents the distance from q to p. Considering a two-dimensional space, there are no other object points inside the circle in which q is center and $Dis(q,p)$ is radius.

The kNN querying principle is interpreted this way: Assume that we have a spatial dataset S. k is an integer, and q is an arbitrary query point. A kNN query finds k points in S such that the distance from these points to q is shorter than distance from other points in S to q, then the kNN query of q is formally defined as shown in equation (2):

$$kNN(q) = \{p_1, p_2, \ldots, p_k\} \tag{2}$$

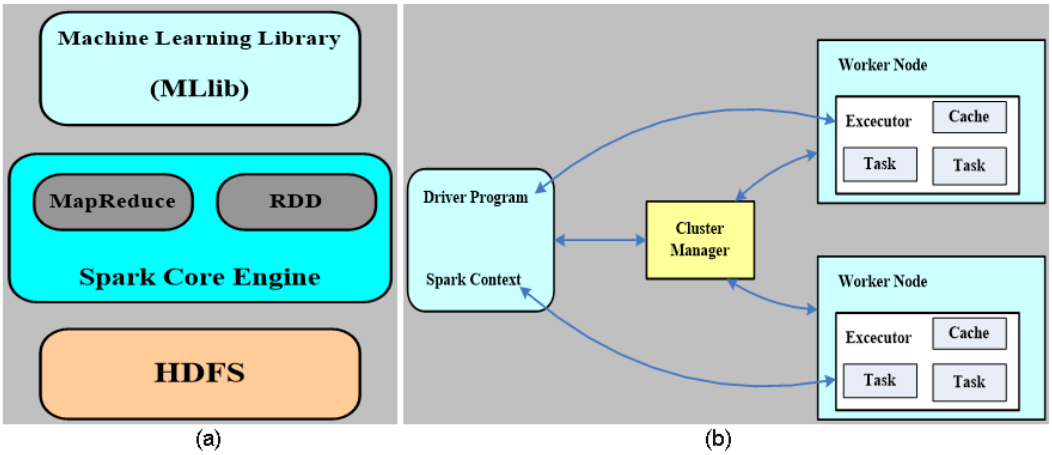
In equation (2), $p_i \in S$, i = 1, 2,..., k. Then for $\forall p \in S$ and $p \notin kNN(q)$, it can be deduced that $Dis(q,p_i) \leq Dis(q,p)$, $Dis(q,p)$ represents the distance between q and p.

The reversed nearest neighbor (RkNN) is explained as follows (Wu et al., 2008): Assume that we have a set of objects p and q is a query point. RkNN is to search for subset $RNN(q)$ of p, formally given as shown in equation (3):

$$RNN(q) = \{p \in P \mid \forall r \in P, Dis(p,q) \leq Dis(p,r) \tag{3}$$

In this equation, $Dis(q,p)$ denotes the distance from q to p. Generally, $r \in NN(q)$ cannot be deduced from $r \in RNN(q)$; that is, there is no necessary connection between them, and vice versa.

Figure 1. Overview of the Spark framework: A) Spark architecture (Yu et al., 2019), B) cluster deployment



Voronoi Diagram

Voronoi diagram (VD) plays the role as a very effective tool in computing geometries. In recent years, VDs have been widely used in spatial databases to describe spatial neighbor relationships; they also are used for realization of a spatial neighbor query, spatial interpolation, and buffer analysis (Moutafis et al., 2021).

In Euclidean space, assume we have a set $P = \{p_1, p_2, ..., p_n\}$ composed of target nodes. The node $p_i \in P$ is called a generator node. Each generator node divides the space into unique regions, which are called Voronoi regions, and each Voronoi region does not overlap with each other. A generator point corresponds to a Voronoi region, constituting a space division. The edges between each Voronoi region are called Voronoi edges. Each Voronoi region is surrounded by multiple Voronoi edges, the regions with the same edge are called adjacent Voronoi regions, and their corresponding generator points are called adjacent generator points. With point sets $P = \{p_1, p_2, ..., p_n\}$ as generator points, $i, j \in In = \{1, ..., n\}$. The Voronoi region of generator point pi can then be defined as $VP(p_i) = \{p \mid dist(p, p_i) \leq dist(p, p_j)\}$, where $Dis(p, p_i)$ denotes the minimum distance from p to p_i. Therefore, the VD generated by p is $VD(P) = \{VP(p_1), ..., VP(p_n)\}$. The VD example is illustrated in Figure 2.

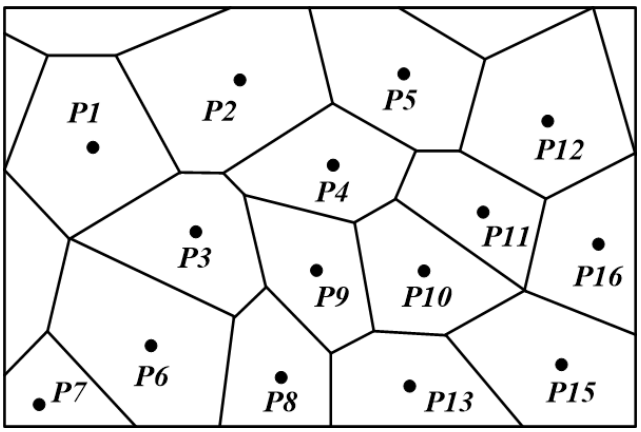
PROPOSED METHOD

For this paper, we proposed a Spark-based RkNN query method. First, we loaded the global grid index of the dataset, and then we could obtain the local Voronoi graph index containing querying point q through retrieval of grid indexes. Second, we loaded local VD indexes and initiated task execution. Finally, we used the VD-based RkNN filtering-refining algorithm in each partition to obtain reversed nearest neighbors of q, and result is stored in HDFS.

Spark-Based Index Construction

To complete the parallel reversed kNN query, we constructed a two-layer indexing structure, including global index as well as the local index. The global index, taking the form of a grid index, is stored in the master node. For Global index, it divides data into various data blocks through grid division. Then local indexes are established on each data block, and the local indexes adopt the Voronoi graph

Figure 2. Voronoi diagram example (Suri & Verbeek, 2016)



index structure. These data are stored in each worker node. The construction process of the double-layer indexing of Spark-based Grid-VD is shown in Figure 3.

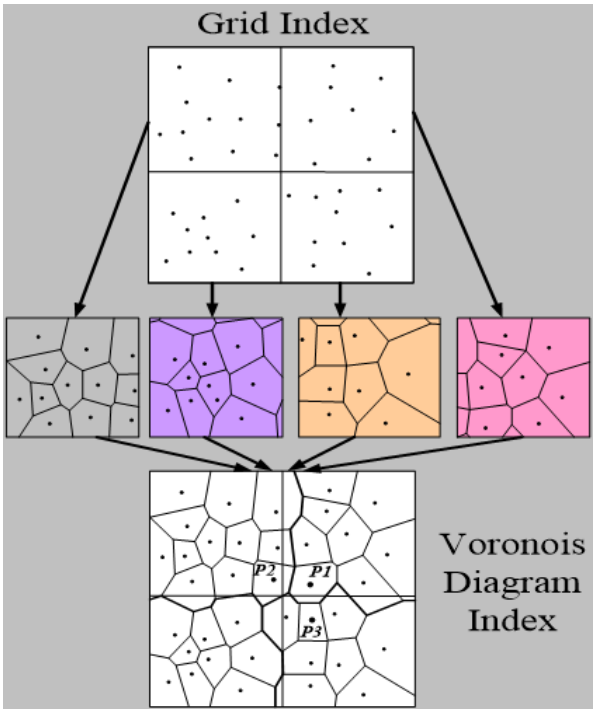
First, the data of each partition of dataRDD is sampled, 1% of the data is selected, and these data are sent to the master node to generate the grid indexes GI. Then the data in each partition is allocated to the corresponding grids by using the grid indexes. For a data point in each partition, if the data point is included in a certain grid, it will be assigned to this grid, and the allocation result will generate a new grid partition RDD, GP-RDD. Then the data with the same grid ID in GP-RDD will be reallocated to a new partition; that is, to perform the repartition, and for objects in the new partition, a Voronoi index will be created to form a VI-RDD.

Filtering-Refining Algorithms Based on RkNN

Given the VD $VD(P)$ of the dataset P, querying point q, the improved RkNN algorithm based on VD includes two steps of filtering and refining. First, the candidates that may become the results are obtained by the filtering step, positions of q are determined in $VD(P)$, and we divide space into six equal areas according q. For each region, first k-th level adjacency generator points of q are put into the candidate set. Algorithm 1 explains the filtering process.

Algorithm 1 RkNN-Based Filtering Algorithm
Input: Query point q, VD $VD(P)$ of the dataset P, k value of the RkNN.
Output: Candidate set $S_c(i)$
Start
For i = 1 to 6 **do**
$S_c(i) \leftarrow \varnothing$

Figure 3. Double-layer indexing structure



End for
Determine positions of q in $VD(P)$
Divide P into six regions
For each $S_c(i)$ do
For i = 1 to k do
Place the i-th adjacent generator points of q into S_c
End for
Return $S_c(i)$
End

Next, through the refining step, the candidate points that cannot be the result are removed from the candidate set. The k-th closest neighbor of each point p in the candidate set is calculated. If the distance from p to the k-th closest neighbor is smaller than the distance from p to q, then p will be deleted from the candidate set. At last, candidate sets of different regions are merged to get final results. Algorithm 2 gives the refined algorithm.

Algorithm 2 RkNN-Based Refining Algorithm
Input: Query point q, VD $VD(P)$ of the dataset P, k value of the RkNN, candidate set S_c from Algorithm 1
Output: RkNN result
Start
$S_c \leftarrow \varnothing$;
For each point p in $S_c(i)$ do
pk = k-th closest neighbor of p
If $Dis(p,q) > Dis(p,p_k)$ then
$S_c \leftarrow Sc(i) - p$
End for
$S_c \leftarrow Sc \cup S_c(i)$
$result \leftarrow S_c$
End

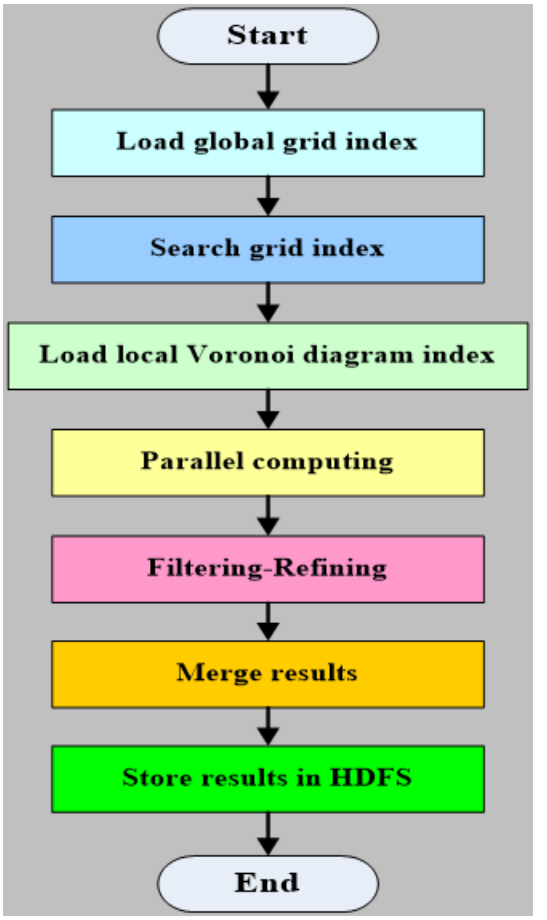
The refinement step will not delete true results and will not return results that are not actual RkNNs. The proposed method first processes the partition where the object in the candidate set is located. Then the refining algorithm is executed to process all objects from the candidate set, and the correctness of the results is guaranteed by removing false candidates that are unlikely to be true results.

Parallel RkNN Query Based on Spark

The basic idea of the proposed Spark-based paralleled RkNN query algorithm is as follows: The algorithm includes two steps, filtering and refining.

Given the RDD of the double-layer index structure, the algorithm first queries the global grid index, locates the grid according to querying q, determines corresponding local index, and performs the filtering and refining step in the partition where the local index RDD is located. In the process of executing the filtering algorithm, if the points that belong to the candidate set are in other adjacent partitions, the corresponding partitions need to be processed in parallel, the filtering algorithm is executed again, and the candidate sets are merged to have the final candidate set. Then, the refining algorithm is executed in the partition where the candidate set is located, and the final parallel RkNN result is calculated. The diagram is illustrated in Figure 4.

Figure 4. Diagram of proposed method



EXPERIMENTS AND ANALYSIS

In the experiment, we compared two algorithms, MRSLICE (García-García et al., 2019) and RkNN-LS (García-García et al., 2017), with our algorithm. The influence of selection of k value on the overall result, the impact of the scale of data objects and quantity of query objects on the running time, the influence of the density of data objects on the amount of candidate points, and the impart of the number of queries on the accuracy of the algorithms were tested, respectively.

We performed the experiment on a Spark distributed cluster containing four nodes (one master node and three worker nodes). The details of the experimental configuration are shown in Table 1. The experiment dataset was composed of real-world data, and the datasets were properly adjusted and optimized in the experiments. The dataset was the San Joaquin County Road network, with a total of 29,306 nodes, and the data include node IDs, standardized X coordinates, and standardized Y coordinates (https://users.cs.utah.edu/~lifeifei/research/tpq/SF.cnode). The results were obtained from the mean values of 100 repeated queries and then compared with the performance of different algorithms.

First, we analyzed the impact of the change of k on the querying time of the MRSLICE algorithm (García-García et al., 2019), the RkNN-LS algorithm (García-García et al., 2017), and the proposed algorithm, and the quantity of query points was 10. Because the value of k keeps

Table 1. Experimental configurations

Item	Configuration
CPU	Intel Xeon E5 2.3 GHz
Memory	32 GB
Hard disk	2 TB
Ethernet card	Broadcom 5720 QP 1 GB
Ethernet switch	Huawei S3700
Operating system	CentOS 6.4
Hadoop version	Hadoop-2.7.1 Spark-2.3.1
JDK version	JDK 1.8

increasing, querying times of the three methods are on the rise. In the experiment, we selected k = 5, 10, 15, and 20, and Figure 5 illustrates the comparison of the querying time of different algorithms with different values of k. Note that in Figure 5 the impact of the change of k value on the proposed algorithm and the RkNN-LS algorithm is obviously less than that of the change of k value on the MRSLICE algorithm. The reason for this difference is that when the value of k rises, the quantity of candidates will be increased, and the amount of data processed by each subnode increases. The MRSLICE algorithm is based on Spatial Hadoop and adopts a disk storage strategy. Reading a large number of candidate sets requires multiple disk accesses, increasing the cost of input and output operations. The proposed algorithm and the RkNN-LS algorithm are based on Spark, and adopt a memory-based computing strategy, which reduces the disk access time, and the data used in the query process is directly available, alleviating the impact of k values on execution time. Compared with the proposed algorithm, the RkNN-LS algorithm needs to execute the kNN query (K is much larger than k), and when the k value increases, the quantity of candidates will be raised, whereas for the proposed algorithm, the number of query accesses of the k-th NN of each candidate point is $6\times k$, so as the value of k increases, the performance of the proposed algorithm is better than RkNN-LS.

Next, we analyzed the impact of the quantity of data objects in the dataset on the querying time. The nodes in the object set were randomly selected as the query objects. The number of data objects were set as 1,000, 2,000, 3,000, 4,000, 5,000, 6,000, and 7,000, respectively, and the results are shown in Figure 6. We noticed that as the scale of the data point set becomes larger, the query times of the three algorithms all showed an upward trend. The MRSLICE algorithm has no data point reduction and pruning process, so its performance was the lowest. When the RkNN-LS algorithm calculated the search area, it did not perform parallel reduction processing on the data point set, so as the number of data points surged, the calculation process costs longer. The proposed algorithm first performed parallel filtering and refining processing on the data point set and used the characteristics of the VD to reduce a large number of non-candidate points. As the data point set became larger, it had little effect on the performance of the proposed algorithm. Therefore, as in Figure 6, the upward trend of the proposed algorithm is relatively flat. In summary, the proposed algorithm surpasses other algorithms in reducing data point sets.

The experimental conditions remained the same, and we further analyzed the impact of the change in the quantity of data points in the dataset on the I/O cost. The results are shown in Figure 7. When number of data points was gradually increasing, the proposed algorithm had an average improvement of 20.2% and 42.1%, compared with MRSLICE and RkNN-LS algorithms, respectively.

Figure 5. Impact of k values on the query times of different algorithms

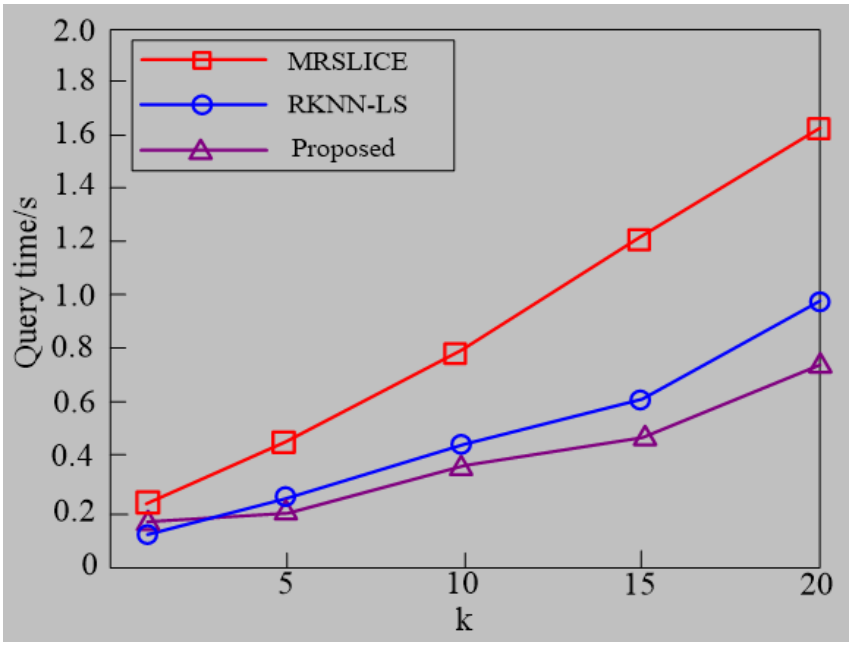
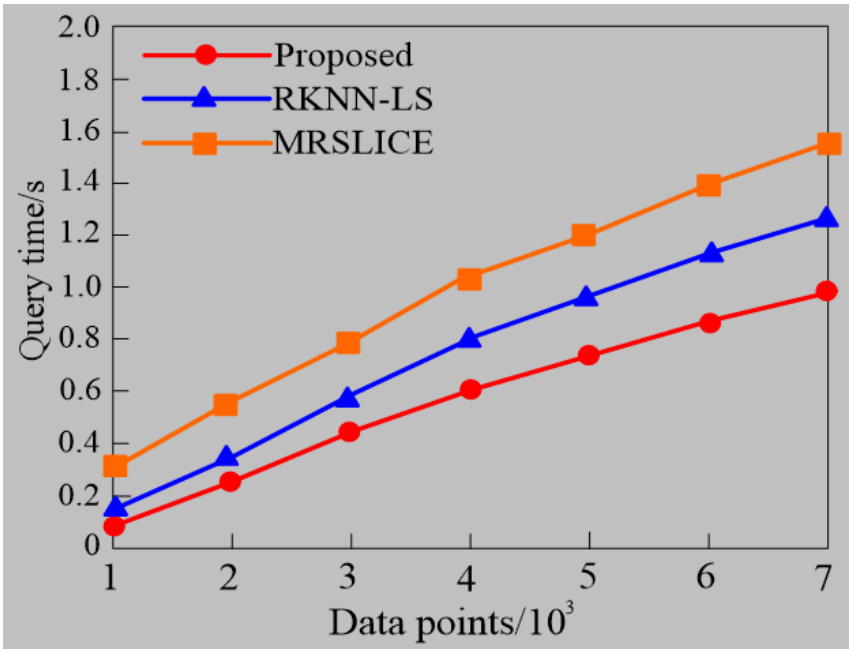
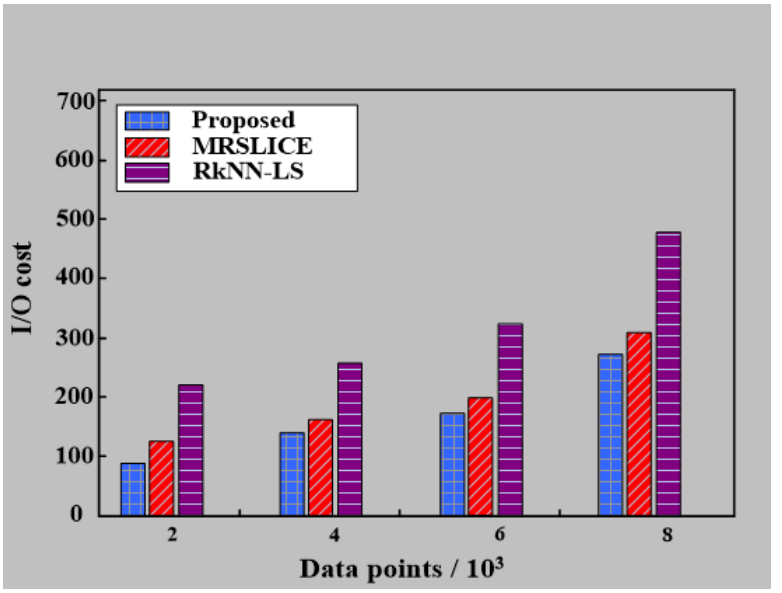


Figure 6. Effects of dataset scale on query time



Then, we analyzed the impact of the quantity of query objects on the query time. Figure 8 shows the influence of the quantity of query points on querying time. The experiment set the k as 10, the number of querying objects as 2, 4, 6, 8, 10, 16, 24, and 32, respectively. Note that the

Figure 7. Effects of dataset scale on I/O cost



querying time of the three algorithms gradually increased when the number of query points rose. Among them, the query time of the MRSLICE algorithm had an obvious upward trend because it needed to calculate its RkNN for each query point. As the number of query points surged exponentially, the querying time used by the algorithm also increased exponentially. However, the rising trend of the query time of the RkNN-LS and the proposed algorithm was relatively gentle because both algorithms had the process of processing the query point set, so when the number of querying points surged exponentially, performance was not greatly affected. In the stage of optimizing the query point set, the time complexity of the proposed algorithm was lower than that of the RkNN-LS. Therefore, when the quantity of query objects gradually increased, the proposed algorithm had an average performance improvement of 19.7% and 38.9%, compared with the MRSLICE and RkNN-LS algorithms, respectively.

Finally, we tested the accuracy of the algorithms. We kept other conditions the same as in the experiment, and the accuracy of the three algorithms was tested when the quantity of queries was set to 20, 40, 60, 80, 100, 120, and 140, respectively. The final result took the average value. As shown in Figure 9, when the quantity of queries gradually grows, the accuracy of the proposed algorithm was higher than the other two algorithms, proving that only useless data points were removed in the filtering and refining steps of the proposed algorithm, ensuring the correctness of the results.

CONCLUSION

For this paper, we studied the parallel RkNN querying method based on the Spark framework, and we used the good characteristics of the VD to construct a two-layered indexing structure based on the grid and the VD. We proposed a filtering-refining algorithm based on RkNN to further remove useless data points and speed up the query under the premise of ensuring parallel processing efficiency. We carried out experiments on real-world datasets and compared our method with the LocationSpark-based RkNN-LS algorithm and the SpatialHadoop-based MRSLICE algorithm. The experiment results verified that in comparison with other algorithms, the proposed method exhibited better query

Figure 8. Effects of different quantities of querying object on the querying time

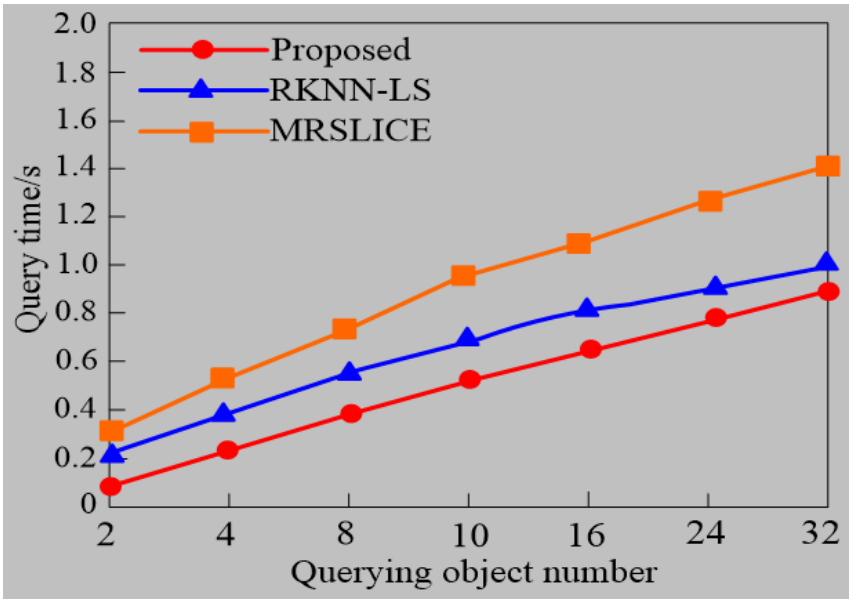
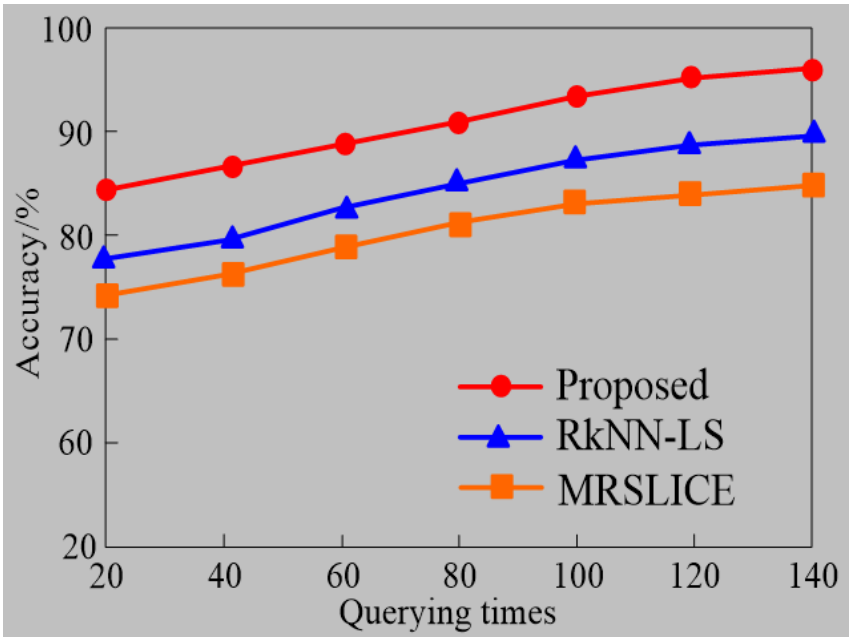


Figure 9. Effects of querying times on the accuracy performance



performance and excellent stability. In the next step, we plan to study the spatial join query based on the Spark framework and further improve the system performance through the improvement of the indexing structure.

DATA AVAILABILITY

The data used to support the findings of this study are included within the article.

CONFLICTS OF INTEREST

The authors declare that there is no conflict of interest regarding the publication of this paper.

FUNDING STATEMENT

This work was supported by research on the consistency checking algorithm of spatial direction relation in spatial database (No. 232102210085) and research on key technologies of the authentication mechanism based on enhanced blockchain framework (No. 232102210197).

REFERENCES

Baharin, S. S. K., & Akunne, P. O. (2020). Analysis of spatial database performance for location intelligence. *Journal of Advanced Computing Technology and Application*, *2*(2), 1–8. https://jacta.utem.edu.my/jacta/article/view/5229

Breunig, M., Bradley, P. E., Jahn, M., Kuper, P., Mazroob, N., Rosch, N., Al-Doori, M., Stefanakis, E., & Jadidi, M. (2020). Geospatial data management research: Progress and future directions. *ISPRS International Journal of Geo-Information*, *9*(2), 95. doi:10.3390/ijgi9020095

Chen, Y., Zhou, L., Tang, Y., Singh, J. P., Bouguila, N., Wang, C., Wang, H., & Du, J. (2019). Fast neighbor search by using revised kd tree. *Information Sciences*, *472*, 145–162. doi:10.1016/j.ins.2018.09.012

Choi, Y., Baek, J., & Park, S. (2020). Review of GIS-based applications for mining: Planning, operation, and environmental management. *Applied Sciences (Basel, Switzerland)*, *10*(7), 2266. doi:10.3390/app10072266

Dhanabal, S., & Chandramathi, S. (2011). A review of various k-nearest neighbor query processing techniques. *International Journal of Computer Applications*, *31*(7), 14–22.

García-García, F., Corral, A., Iribarne, L., & Vasssilakopoulos, M. (2017). RkNN query processing in distributed spatial infrastructures: A performance study. In Y. Ouhammou, M. Ivanovic, A. Abelló, & L. Bellatreche (Eds.), *Model and data engineering. MEDI 2017* (pp. 200–207). Springer. doi:10.1007/978-3-319-66854-3_15

García-García, F., Corral, A., Iribarne, L., & Vasssilakopoulos, M. (2019). MRSLICE: Efficient RkNN query processing in SpatialHadoop. In K. D. Schewe & N. Singh (Eds.), *Model and data engineering. MEDI 2019* (pp. 235–250). Springer. doi:10.1007/978-3-030-32065-2_17

García-García, F., Corral, A., Iribarne, L., Vasssilakopoulos, M., & Manolopoulos, Y. (2020). Efficient distance join query processing in distributed spatial data management systems. *Information Sciences*, *512*, 985–1008. doi:10.1016/j.ins.2019.10.030

Grolinger, K., Hayes, M., Higashino, W. A., L'Heureux, A., Allison, D. S., & Capretz, M. A. M. (2014). Challenges for MaPreduce in big data. In *Proceedings of the 2014 IEEE World Congress on Services*. IEEE. doi:10.1109/SERVICES.2014.41

Ji, C., Qu, W., Li, Z., Xu, Y., Li, Y., & Wu, J. (2015). Scalable multi-dimensional RNN query processing. *Concurrency and Computation*, *27*(16), 4156–4171. doi:10.1002/cpe.3513

Jiang, X., Ma, J., Jiang, J., & Guo, X. (2019). Robust feature matching using spatial clustering with heavy outliers. *IEEE Transactions on Image Processing*, *29*, 736–746. doi:10.1109/TIP.2019.2934572

Li, R., Wang, R., Liu, J., Yu, Z., He, H., He, T., Ruan, S., Bao, J., Chen, C., Gu, F., Hong, L., & Zheng, Y. (2021). Distributed spatio-temporal k nearest neighbors join. In *SIGSPATIAL '21: Proceedings of the 29th International Conference on Advances in Geographic Information Systems*. Association for Computing Machinery. doi:10.1145/3474717.3484209

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D. B., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M. J., Zadeh, R., Zaharia, M., & Tawalkar, A. (2016). MLlib: Machine learning in Apache Spark. *Journal of Machine Learning Research*, *17*(1), 1235–1241.

Moutafis, P., Mavrommatis, G., Vassilakopoulos, M., & Corral, A. (2021). Efficient group K nearest-neighbor spatial query processing in Apache Spark. *ISPRS International Journal of Geo-Information*, *10*(11), 763–775. doi:10.3390/ijgi10110763

Pant, N., Fouladgar, M., Elmasri, R., & Jitkajornwanich, K. (2018). A survey of spatio-temporal database research. In N. Nguyen, D. Hoang, T. P. Hong, H. Pham, & B. Trawiński (Eds.), *Intelligent information and database systems. ACIIDS 2018* (pp. 115–126). Springer. doi:10.1007/978-3-319-75420-8_11

Qiao, B., Hu, B., Zhu, J., Wu, G., Giraud-Carrier, C., & Wang, G. (2020). A top-k spatial join querying processing algorithm based on spark. *Information Systems*, *87*, 101419. doi:10.1016/j.is.2019.101419

Suri, S., & Verbeek, K. (2016). On the most likely Voronoi diagram and nearest neighbor searching. *International Journal of Computational Geometry & Applications, 26*(3-4), 151–166. 10.1142/S0218195916600025

Sveen, A. F. (2019). Efficient storage of heterogeneous geospatial data in spatial databases. *Journal of Big Data*, *6*(1), 1–14. doi:10.1186/s40537-019-0262-8

Ventocilla, E. (2019). Big data programming with Apache Spark (171–194). In A. Said & V. Torra (Eds.), *Data science in practice*. Springer. doi:10.1007/978-3-319-97556-6_10

Wan, S., Zhao, Y., Wang, T., Gu, Z., Abbasi, Q. H., & Choo, K.-K. R. (2019). Multi-dimensional data indexing and range query processing via Voronoi diagram for internet of things. *Future Generation Computer Systems*, *91*, 382–391. doi:10.1016/j.future.2018.08.007

Wang, H., Fu, X., Xu, J., & Lu, H. (2019). Learned index for spatial queries. In *Proceedings of the 2019 20th IEEE International Conference on Mobile Data Management (MDM)*. IEEE. doi:10.1109/MDM.2019.00121

Wang, M., Xu, X., Yue, Q., & Wang, Y. (2021). *A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search.* arXiv:2101.12631 [cs.JR].

Wu, W., Yang, F., Chan, C.-Y., & Tan, K.-T. (2008). FINCH: Evaluating reverse k-nearest-neighbor queries on location data. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *1*(1), 1056–1067. doi:10.14778/1453856.1453970

Xia, H., Liu, Z., Efremochkina, M., Liu, X., & Lin, C. (2022). Study on city digital twin technologies for sustainable smart city design: A review and bibliometric analysis of geographic information system and building information modeling integration. *Sustainable Cities and Society*, *84*, 104009. doi:10.1016/j.scs.2022.104009

Xu, J., Zhao, Y., & Yu, G. (2021). An evaluation and query algorithm for the influence of spatial location based on RkNN. *Frontiers of Computer Science*, *15*(2), 1–9. doi:10.1007/s11704-020-9238-2

You, S., Zhang, J., & Gruenwald, L. (2015). Large-scale spatial join query processing in cloud. In *Proceedings of the 2015 31st IEEE International Conference on Data Engineering Workshops.* IEEE. doi:10.1109/ICDEW.2015.7129541

Yu, J., Wu, J., & Sarwat, M. (2015). Geospark: A cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL '15: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. Association for Computing Machinery. doi:10.1145/2820783.2820860

Yu, J., Zhang, Z., & Sarwat, M. (2019). Spatial data management in Apache Spark: The geospark perspective and beyond. *GeoInformatica*, *23*(1), 37–78. doi:10.1007/s10707-018-0330-9

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., & Franklin, M. J. (2012). Fast and interactive analytics over Hadoop data with Spark. *Usenix, 37*(4), 45–51.

Zhang, P., Zhang, D., Yang, Y., Zhang, W., Wang, Y., Pan, Y., & Liu, X. (2022). A case study on integrated modeling of spatial information of a complex geological body. *Lithosphere, 2022*(Special 10), 2918401. 10.2113/2022/2918401