Proceedings of the 6th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/68/20241385

Optimization study of pruning strategy based on KNN trajectory similarity query

Jingru Zhu[1,3,*], Zhongyu Wang[2]

[1]Department of Information Science and Engineering, Ocean University of China, Shandong, China
[2]Department of Information Science and Engineering, Ocean University of China, Shandong, China

[3]3563779233@qq.com
*corresponding author

Abstract. With the development of GPS positioning technology, a large amount of spatiotemporal trajectory data has been generated. Due to the complex structure of trajectory data, which has irregular spatial shapes and continuous temporal sequence attributes, querying massive trajectory data poses certain challenges. The pruning strategy of existing trajectory similarity query methods is not very effective. Even after pruning operations, there are still a large number of trajectories that need to undergo distance calculations to confirm whether they are similar trajectories. This paper proposes multiple local pruning optimization schemes, which maximally reduce the number of trajectories in the candidate set. Specifically, it starts with region pruning in the index space, eliminating index spaces with distances greater than the maximum similarity distance from the query trajectory. Then, it performs distance pruning between trajectories, removing trajectories that do not meet the distance conditions. Finally, it adds a termination condition: when the distance between the current index space and the query trajectory is greater than the maximum similarity distance and the number of trajectories in the result set is K, the loop is exited, and the query process ends. Tests on real datasets demonstrate that the KTSS method outperforms current algorithms of the same type.

Keywords: Trajectory Data Mining, Trajectory Similarity, Spatiotemporal Big Data.

1. Introduction

With the popularity of mobile devices and the development of location-based services, a large amount of spatiotemporal trajectory data has been generated. Similarity queries involve searching for trajectories similar to a given trajectory in massive datasets. This can help us understand the behavioral patterns and trends of moving objects, and provide important references for fields such as traffic management and urban planning.

In today's society, with the popularity of mobile devices and location technology, there has been explosive growth in the accumulation of large-scale trajectory data. These trajectory data continuously cover various aspects such as human mobility, traffic flow [1], logistics distribution [2], etc., providing valuable information resources for urban management [3], traffic planning [4], intelligent transportation systems [5], and other fields.

DOI: 10.54254/2755-2721/68/20241385

Similarity queries involve searching for trajectories similar to a given trajectory in massive datasets, which is important for multiple fields. Firstly, as cities expand and population mobility increases, traffic management becomes more complex. Similarity queries can help better understand and optimize traffic flow, improving urban traffic efficiency. Secondly, for the logistics industry, trajectory similarity queries can achieve more intelligent and efficient distribution route planning, reducing logistics costs and improving logistics efficiency. In addition, for personalized recommendations and location-based services, precise similarity queries can provide users with information and services that better meet their actual needs, enhancing user experience.

Due to the large number of trajectories and the complexity of trajectory similarity measurement, it is extremely time-consuming and resource-intensive to find the top K trajectories similar to a given trajectory by extracting all data from the database for trajectory similarity measurement one by one. Key-value data storage systems such as HBase [6] are widely used in big data management because they can achieve fast read/write throughput and fast lookup of row keys. However, most trajectory similarity query methods currently have overly crude pruning strategies, leaving too many remaining trajectories in the candidate set, making trajectory similarity search queries inefficient.

To address the above-mentioned problems, this paper proposes several pruning strategy optimization methods based on KNN trajectory similarity queries to conduct KNN similarity trajectory queries more quickly. Firstly, for the input query trajectory q, preprocessing is performed. The spatial index representation of the trajectory is generated through the XZ* [8] indexing mechanism, and the trajectory is simplified using the Douglas-Peucker algorithm. The combination of the two generates the key value Key in the trajectory storage structure. When performing similarity trajectory queries, the impossible index spaces that may exist with the query trajectory are first pruned through global pruning. Then, local pruning is used to remove dissimilar trajectories from the remaining index spaces. Finally, the DTW algorithm is used to calculate the distance between the query trajectory and the current retrieval trajectory. The trajectories with the shortest distances to the query trajectory are retained and sorted according to this distance, forming a collection of K trajectories closest to the query trajectory as the result of a KNN trajectory similarity query. The innovations of this paper are summarized as follows:

(1) This work proposes a KNN trajectory similarity query model, KTSS, to improve the efficiency of KNN similarity queries. When performing region pruning operations, the physical position of the index space is considered. The characteristics of the DTW algorithm are applied to pruning operations, and a termination condition is added to shorten the query time.

(2) The model is verified on real datasets. Through result comparison analysis, the KTSS model is slightly better than current state-of-the-art algorithms.

2. Definitions

Definition 1 (Trajectory): A trajectory is composed of multiple trajectory points, denoted as $t = p_1, \ldots, p_n$. Each trajectory point p consists of latitude and longitude, $p_i = (lat, lng)$;p_1 is the starting point, and p_n is the endpoint.

Definition 2 (Trajectory Similarity Measurement Method): This paper utilizes the Dynamic Time Warping (DTW) algorithm to measure the similarity between two trajectory sequences. The fundamental idea of DTW [7] is to dynamically adjust the correspondence between two trajectory sequences to achieve one-to-one matching, calculating the cumulative distance between corresponding points as the measure of their similarity. This is represented as:

$$D[i][j] = d(a_i, b_j) + \min(D[i-1][j], D[i][j-1], D[i-1][j-1]) \quad (1)$$

Definition 3 (Minimum Bounding Rectangle of a Trajectory): In this paper, the smallest area, parallel to the coordinate axes, that contains all the points of a trajectory t, is referred to as the Minimum Bounding Rectangle (MBR).

Definition 4 (Element): Includes root space and all its subspaces, and the subspaces of those subspaces; each individual region is referred to as an element.

DOI: 10.54254/2755-2721/68/20241385

Definition 5 (Enlarged Element): Starting from the bottom-left corner of an element, if the length and width of the element are both doubled, the resultant area is called an Enlarged Element (EE).

Definition 6 ($\mathrm{MinDIS}(q, \mathrm{IS})$) This calculates the minimum possible distance between the Index Spaces (IS) and the query trajectory q. It is calculated as follows, where p is a point on the edge of the MBR of the query trajectory, and $d(p, \mathrm{EE})$calculates the distance between p and the Enlarged Element.

$$\mathrm{MinDIS}(q, \mathrm{IS}) = \max_{p \in q.\mathrm{MBR} \wedge q} d(p, \mathrm{IS}) \tag{2}$$

Problem Definition (KNN Trajectory Similarity Query Problem): The task is to find the k trajectories closest to a query trajectory from a dataset of trajectories, which constitutes the KNN Trajectory Similarity Query Problem. In symbolic language: Given a set of trajectories T, a query trajectory q, a positive integer k, and a distance function f, the KNN trajectory similarity query returns a subset T' of trajectories from T, containing k trajectories. The query result satisfies the following condition:

$$\mathrm{KNN\_Quary}(T, q, k, f) = \{t_i \in T' | \forall t_j \in T \wedge t_j \notin T', f(t_i, q) < f(t_j, q)\} \tag{3}$$

Here, the length of the trajectory set T' is k, and all trajectories in T' are included in the trajectory set T. The function $f(t_i, q)$ calculates the Euclidean distance between a point and a trajectory.

3. Framework

Key-Value databases like HBase, commonly used for storing large-scale data, utilize key-value pairs to store objects. However, in the dataset used in this paper, each trajectory consists of multiple trajectory points. Therefore, appropriate key-value allocations are required for the trajectory data objects. This paper employs the XZ* index to represent trajectories, utilizing a bijection function to generate index values for key-value pairs in trajectory storage. In real-life scenarios, a trajectory data may contain multiple trajectory points, and many of these points may be closely spaced. Redundant computations for approximate duplicates consume time and waste computational resources. Hence, this paper employs the Douglas-Peucker (DP) [9] algorithm to compress trajectory points as much as possible without compromising accuracy, for trajectory simplification. Concurrently, multiple pruning methods are combined for KNN trajectory similarity queries to shorten query times and enhance query efficiency.

4. Pruning Operations

Trajectories are two-dimensional data with spatial forms. The trajectories in the result set of a KNN similarity query are usually similar in shape to the query trajectory. Therefore, starting from the query trajectory itself and using it as the center for comparisons with surrounding trajectories can identify some trajectories that can be pruned directly without needing to calculate the actual distance between them. This involves identifying the necessary conditions for the similarity between the query trajectory and candidate trajectories, eliminating as many unnecessary trajectory computations as possible to improve retrieval efficiency. The following sections detail the pruning operations used in this study.

4.1. Pruning Operation 1

Pruning operation 1: If there exists a trajectory point p in trajectory t_1 such that $d(p, t_2) > \varepsilon$, then trajectories t_1 and t_2 are not similar. Here, $d(p, t_2) = \min_{x \in t_2} d(p, x)$, where $d(p, x)$ calculates the Euclidean distance between the two points, and ε is the maximum similarity distance.

Proof: $f(t_1, t_2) = D_{DTW}(t_1, t_2)$. According to the DTW formula, $D_{DTW}(t_1, t_2) \geq d(p, x)$. Here, the trajectory point x in t_2 is the matching point for point p in trajectory t_1, hence we have $d(p, t_2) \leq d(p, x)$ $D_{DTW}(t_1, t_2)$. Since $d(p, t_2) > \varepsilon$, $f(t_1, t_2) = D_{DTW}(t_1 > \varepsilon)$, thus t_1 and t_2 are not similar.

Example: In trajectory t_1, there is a trajectory point p, which matches with point x in trajectory t_2. We use the DTW method to measure trajectory similarity, requiring each point in one trajectory to match with points in another trajectory for distance calculation. The DTW algorithm finds the optimal path that minimizes the sum of distances between matched points. Since $d(p, x) > \varepsilon$, it is impossible for trajectories t_1 and t_2 to be similar.

DOI: 10.54254/2755-2721/68/20241385

4.2. Pruning Operation 2

Pruning operation 2: Iteratively traverse the index space, calculate the $MinDIS(q, IS)$ value, and if the distance between the query trajectory q and the index space is greater than the maximum similarity distance ε, this index space can be pruned. While pruning, record the position of this index space; if the next index space is completely contained within this recorded position, directly prune this index space without needing to calculate the distance.

Proof: Each enlarged element is divided into four subspaces to form the index space. If the distance $d(IS, q)$ from the index space (Index Space, IS) to the query trajectory $d(IS, q) > \varepsilon$, then all trajectories within the index space are not similar to q. Because at least one trajectory point p exists in the index space such that $d(p, q) \geq d(IS, q)$, and since $d(IS, q) > \varepsilon$, then $f(p, q) \geq d(IS, q) > \varepsilon$, thus all trajectories passing through this space and the index space itself are not similar to q and therefore can be pruned. Save the pruned index space's position, so if an index space to be checked is fully contained within a recorded pruned space, it can be pruned directly without calculation. $d(IS, q)$ calculates the minimum possible distance from the track q to the index space IS, and if the index space to be calculated is included in the recorded spatial location, then its distance from the track q must be greater than the maximum similar distance ε. Because the distance between the included index space to be calculated and the track q is greater than the distance between the recorded spatial position and the track q, and because the distance between the recorded spatial position and the track q is greater than the maximum similarity distance ε, the track in this index space must not be similar to the track q and can be pruned directly.

4.3. Pruning Operation 3

Pruning operation 3: If the query trajectory q is similar to the candidate trajectory t, then the distance d_1 from the start point of q to the start point of t must be less than or equal to the maximum similarity distance ε, and the distance d_2 from the endpoint of q to the endpoint of t must also be less than or equal to ε; the sum d_3 of d_1 and d_2 must also be less than or equal to ε. Otherwise, the query trajectory q and the candidate trajectory t are not similar.

Proof: The DTW algorithm used to measure trajectory similarity features matching trajectory points between two trajectories to find an optimal path that minimizes their total distance. The DTW requires that the starting and ending points of two trajectories correspond one-to-one. Thus, whether it's the distance d_1 between starting points or d_2 between ending points, these will be part of the final DTW result. If d_1 or d_2 exceeds ε, further calculations of distances between other trajectory points are unnecessary, and their total DTW distance will certainly exceed ε. Therefore, if the distance d_1 between starting points or d_2 between endpoints of q and t exceeds the threshold ε, or if $d_3 > \varepsilon$, the two trajectories are definitely not similar and can be pruned directly.

If the distance d_1 between the start point of the track q and the candidate track t, or the distance d_2 between the end point is greater than the threshold ε, the two tracks are not similar. At the same time, the distance of $d_1 + d_2$ and d_3 must also be less than the threshold ε. According to the previous text, the distance of DTW algorithm consists of the distance sum of matched start and end points, end points and other one-to-one corresponding trajectory points, so d_3 must be in the DTW result, so d_3 must also be less than the threshold ε, so that the track q to be investigated can be similar to the candidate track t. If $d_3 > \varepsilon$, the trajectory q and trajectory t must not be similar, and the trajectory can be pruned directly.

4.4. Pruning Operation 4

Pruning operation 4: During the algorithm process, index spaces are sorted according to their distance to the query trajectory q to form a priority queue, with shorter distances at the front of the queue. Whenever a new index space is retrieved in order from the priority queue, first check if the distance between the query trajectory q and the current index space exceeds the threshold ε. If the number of trajectories in the result set equals k and the distance between q and the index space exceeds the threshold, end the entire query process immediately. Prune all remaining index spaces and stop further comparison queries.

DOI: 10.54254/2755-2721/68/20241385

Proof: The priority queue sorted by the distance between the query trajectory q and the index spaces stores index spaces that have not yet been searched. With k trajectories already in the current result set, if the distance between q and an index space exceeds the maximum similarity distance ε, then the distance of any trajectory point in any trajectory within the index space to q will be greater than or equal to the distance from the index space to q. As the distance from q to the index space is calculated as the minimum possible distance, any trajectory within an index space contained within a recorded space will definitely have a distance to q that is greater than the maximum similarity distance ε. Thus, all trajectories within this index space are not similar to q and can be pruned directly.

5. Experiments

This section conducts extensive experiments on the proposed KNN trajectory similarity query method, comparing and analyzing its performance against advanced trajectory similarity methods to validate the effectiveness of the approach described in this paper.

5.1. Dataset

This work utilized a real-world vehicle trajectory dataset: the T-Drive dataset [10], which includes 7 days of Beijing taxi trajectory data from February 2 to February 8, 2008. Table 1 summarizes the statistical data of the T-Drive dataset.

Table 1. T-Drive Dataset

Dataset	Trajectory Points	Number of Trajectories	Dataset Size
T-Drive	17,662,984	314,086	752MB

5.2. Comparative Methods

The KNN Trajectory Similarity Search framework (KTSS) is compared with other leading methods. Here is a brief introduction to the baseline methods:

DITA [11] is an advanced in-memory trajectory management platform that designs a distributed indexing structure, refines index content, and shares index information among nodes to query trajectory similarity.

DFT [12] stores trajectory data by partitioning between adjacent trajectory points in line segments rather than based on the trajectory itself, reducing data redundancy and enhancing pruning performance.

JUST [13] optimizes spatial and temporal indexes and introduces an efficient indexing structure to accelerate the query and retrieval of spatiotemporal data.

TRASS [8] proposes a trajectory similarity query method based on key-value data storage, leveraging the advantages of key-value data storage combined with an efficient indexing structure to improve retrieval efficiency.

5.3. Experimental Analysis

Table 2 reports the time taken for different methods to perform KNN trajectory similarity searches under various values of k. The method used in this chapter, the KTSS method, is in bold font, while other methods are for baseline comparison. The best-performing method is highlighted in bold black font, and the second best is underlined. DITA and DFT take longer than KTSS because they build many indexes in memory, and each query operation requires locating the trajectory's position using the index, while KTSS directly extracts the trajectory's location. JUST takes longer due to the high number of candidate trajectories requiring many distance calculations. Compared to the method used in this paper, the TRASS algorithm has a more rudimentary local pruning method, resulting in longer query times. As the query number k increases, the query times of all methods also increase but not significantly because each similarity trajectory search starts from the query trajectory itself and expands outward, with many trajectories already returned within the spatial range, so the number of iterations outward only increases slightly. The performance of this chapter's method improves as the value of k increases because more trajectories are returned, and the effectiveness of the local pruning optimization method becomes more

DOI: 10.54254/2755-2721/68/20241385

apparent, improving performance by about 22% compared to the currently best method when the results are optimal.

Table 2. DTW Algorithm KNN Similarity Query Time Table

k	Query Time(ms)				
	DITA	DFT	JUST	TRASS	**KTSS**
50	6622	20723	6750	1774	**1332**
100	6913	21096	7181	2215	**1724**
150	7493	21656	7764	2759	**2165**
200	8289	22155	8604	3537	**2952**

6. Conclusion

This paper proposes a KNN similarity query method for large-scale trajectory data to facilitate faster KNN similarity trajectory queries. Firstly, preprocessing is applied to the input query trajectory q, generating a spatial index representation of the trajectory through the XZ* indexing mechanism. Simultaneously, the DP (Douglas-Peucker) algorithm is used to simplify trajectory features. The combination of these steps generates the key value Key of the trajectory storage structure Key-Value. When conducting similarity trajectory queries, impossible index spaces that could contain trajectories similar to the query trajectory are first pruned through global pruning. Then, local pruning is employed to eliminate dissimilar trajectories in the remaining index spaces. Finally, the DTW (Dynamic Time Warping) algorithm is used to calculate the distance between the query trajectory and the current retrieved trajectories. These distances are sorted, and the top k trajectories with the shortest distances to the query trajectory are retained as the result of a KNN trajectory similarity query. Through various pruning methods, many similarity calculations between the query trajectory and dissimilar trajectories can be omitted. Multiple experiments demonstrate that the performance of KTSS is slightly higher than that of the compared algorithms.

References

[1] J. A. Laval and J.-P. Lebacque, “A review of urban traffic flow models,” Mathematical and computer modelling, vol. 49, no. 3–4, pp. 413–442, 2009.
[2] R. Baldacci, A. Mingozzi, and R. Roberti, “A survey of routing problems in logistics,” European Journal of Operational Research, vol. 202, no. 1, pp. 1–17, 2010.
[3] A. Caragliu, C. Del Bo, and P. Nijkamp, “Smart city: Definitions, dimensions, performance, and initiatives,” Journal of Urban Technology, vol. 18, no. 2, pp. 65–82, 2011.
[4] P. Toth and D. Vigo, “Vehicle routing problems: Formulations and recent advances,” European Journal of Operational Research, vol. 231, no. 1, pp. 1–17, 2013.
[5] T. Dey, M. Quddus, and M. M. Rashid, “Smart urban mobility system: A review,” IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 7, pp. 2969–2993, 2020.
[6] L. George, “Apache HBase: A distributed, scalable big data store,” in ApacheCon US, 2011.
[7] S. Salvador and P. Chan, “Toward accurate dynamic time warping in linear time and space,” Intelligent Data Analysis, vol. 11, no. 5, pp. 561–580, 2007.
[8] H. He et al., “Trass: Efficient trajectory similarity search based on key-value data stores,” in 2022 IEEE 38th international conference on data engineering (ICDE), IEEE, 2022, pp. 2306–2318.
[9] Y. Zheng, “Trajectory data mining: An overview,” ACM Transactions on Intelligent Systems and Technology, vol. 6, no. 3, pp. 1–41, 2015.
[10] J. Yuan, Y. Zheng, X. Xie, and G. Sun, “Driving with knowledge from the physical world,” ACM, p. 316, 2011.
[11] Z. Shang, G. Li, and Z. Bao, “DITA: Distributed in-memory trajectory analytics,” in The 2018 international conference, 2018.

DOI: 10.54254/2755-2721/68/20241385

[12] D. Xie, F. Li, and J. M. Phillips, “Distributed trajectory similarity search,” Proceedings of the Vldb Endowment, vol. 10, no. 11, pp. 1478–1489, 2017.
[13] R. Li et al., “Just: Jd urban spatio-temporal data engine,” in 2020 IEEE 36th international conference on data engineering (ICDE), IEEE, 2020, pp. 1558–1569.