MSBD6000J Spatial and Multimedia Databases

Group 4 Project Report

Topic: Efficient Trajectory Similarity Search

Group Member:
KOO Tin Lok (20344775), LAM Chun Ting, Jeff (12222973)
LO Ngai Hung (20787771), WONG Sing Long, Cyrus (20827870), ZHENG Hao (20797594)

1. Problem Statement

Trajectory data is common in daily life. Huge amount of trajectory data is generated everyday by walking or driving. Trajectory similarity measurement and search is one of the key elements. The problem is how to perform the measurement and search efficiently. In aspect of measurement, for traditional trajectory similarity search, it focuses on semantic similarity search, which finds the trajectory, which is most like the other, or focuses on the shape search, in which is the sub trajectory search. The problem is that for the method mentioned above, it does not focus on the important part of the trajectory, which means it is equally important for each point in the trajectory.

2. Application scenarios

There is a broad range of applications, such as trip planning, traffic analysis, and carpooling. Trajectory similarity search is a problem to find the most similar trajectory in the trajectory database from a query trajectory.

3. Related work

There are numbers of similarity measures, old methods which are more noise sensitive such as Euclidean based [1], Dynamic Time Warping [2] to newer, which is more robust and accurate such as Edit Distance on Real Sequence [3], and Longest Common Subsequence [4]. And For user-oriented similarity search, longest Common Subsequence is chosen. And there are some papers related to data mining techniques to perform a similar effect on user-oriented trajectory search, and those are performing semantic search.

In recent years, machine learning techniques also applied to trajectory similarity search. For example, “DISON” [5] and “NEUTRAJ” [6]. However, those proposed methods are targeted for specific applications or needs. Therefore, a method with efficiency and generality is needed.

4. New solutions proposed and their relationship

4.1. User Oriented Trajectory Similarity Search (SIGKDD, 2012)

By introducing a weight to each trajectory query point, it will indicate the importance of a trajectory, such that it can provide flexible and highly complex query patterns. [7]

<u>Problem Modeling for user-oriented trajectory similarity search</u>
p = <longitude, latitude> point definition
T = {p1, p2, ... , pn} data trajectory
DB = {T1, T2, …, TN} database storing the trajectories
Q = {q1, q2, …., qn} query trajectory, where q is a point

Matching criteria
|loni - lonj| < threshold and |lati - latj| < threshold

Longest Common Subsequence (LCSS) method is used for trajectory search

Heaviest Common Subsequence (HCSS) is defined from this paper, weighted sum of their longest common sequence.

HCSS(T, Q). The specific value can be derived from the following recursive computation:

$$\begin{cases} 0 & \text{if } n = 0 \text{ or } m = 0, \\ w_1 + HCSS(Rest(T), Rest(Q)) & \text{if } p_1,\ q_1 \text{ are matched}, \\ max \begin{cases} HCSS(Rest(T), Q), \\ HCSS(T, Rest(Q)) \end{cases} & \text{otherwise} \end{cases}$$

where $Rest(.)$ denotes the rest part of a trajectory with the first sample point removed.

Return the one with the largest HCSS value.

Optimization in finding computation

The cost of a naive solution is heavy; therefore, filter and refinement steps are taken.

Filter

In this stage, it mainly pruned the database trajectory that the trajectory point does not overlap with the query trajectory estimated area in order to improve the efficiency in user-oriented trajectory search.

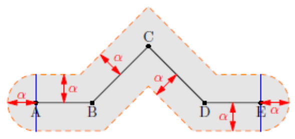


Figure 4: $L = \{A, B, C, D, E\}$ **and** $\mathcal{D}_\alpha(L)$

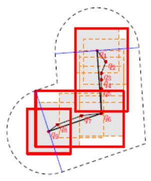


Figure 5: An example of grouped areas

By introducing alpha bounding, the grouping of consecutive query points, and R-Tree indexing, the algorithm can minimize the white area, as well as reduce as much overlapping area (grey area) as the algorithm can. And therefore, we can only consider the trajectory with the point in the red bounding rectangle in the database.

Refinement

Heaviest grouped subsequence is introduced, which act as the upper bound of HCSS in problem finding. k be the number of grouped consecutive points (mentioned above), and with a corresponding weight. For example, in the top right bounding box in figure five, there are 5 points bounded. Then G(q1, q5) will be k*w, where k is 5, and w can be any user defined weights. HGSS will be the sum of that, which will be the HCSS upper bound.

Definition 4. ***(HGSS)*** For any T in the trajectory database, given Q and the corresponding grouped areas $\{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_K\}$, we define the *Heaviest Grouped Subsequence* between them

$$HGSS(T, Q) = \sum_{i=1}^{K} F(i)$$

where $F(i) = \mathcal{G}_i.\mathcal{W}$ if T has at least one sample points in grouped area $\mathcal{G}_i$, otherwise $F(i) = 0$.

For the computed HCSS of a trajectory, if there does not exist a HGSS that is greater than HCSS, then we can just simply prune the rest of the trajectory. And that HCSS trajectory will be the most similar trajectory in the user-oriented trajectory similarity search.

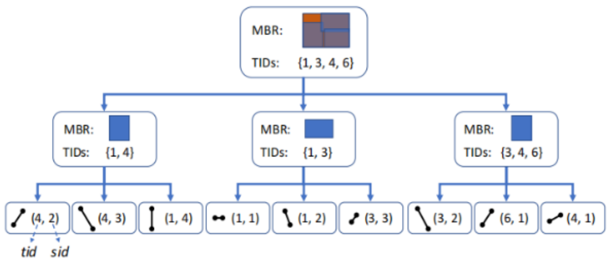
4.2. Distributed Trajectory Similarity Search (VLDB, 2017)

Aiming at improving trajectories similarity search efficiency under enormous trajectory data, this paper designed a distributed trajectory similarity search framework(DFT) and implemented it in Spark, This framework is based on a distributed index structure and some high-performance pruning techniques[8].

As for the Index structure, this paper uses R-tree to construct an index and partition and index trajectories by processing MBR for each of its segments. It represents each segment with its centroid and uses these centroids to build a R-tree.

The whole indexing procedure contains three parts: Partition, local indexing and global indexing.

When partitioning, the framework extracts segments from input trajectories and partitions them according to spatial location. This paper uses Sort-Tile-Recursive partitioning strategy to perform the partitioning. The second phase is local indexing. Each partition shapes like R-tree, except each non-leaf node has all the trajectories' id which appear in its subtree and the leaf node' s representation. It uses 2 points and a tuple(tid,sid) to index a segment l from a trajectory T, where points are the beginning and end of the segment, tid is the trajectory id of T and sid is the position indicator that l in which part of T



In the last phase, global indexing, the master will collect statistics from the worker node and build a global index. It helps the pruning process.

As for the searching procedure, It can be divided into 2 phases: pruning and selecting. There are two kinds of pruning. The first kind is in global indexing. It obtains a pruning bound and, it searches the global indexing and gets partitions which can be pruned by calculating the mindist(Q, A). The second kind is performed at the remaining local index. If the index' s MBR is out of range, trajectories whose segments are inside this MBR can be pruned. At the selection phase, it reconstructs the trajectories that are not pruned, and calculate the distance, sort and find the nearest k trajectories.

The implementation on spark is as follows. The IPartition is to represent a local index, and each local index sends its meta information to the master node, also the global index. The TrajSeg class is to represent the leaf node of a local index, representing a segment of a trajectory. As for the internal node of the index, it should contain a trajectory id set. Here to save memory consumption, this paper compares three methods and adopts a roaring bitmap to store trajectory id sets. As for the overhead of reconstructing trajectory, this paper designs a dual indexing. That is, each partition can be seen as a local index and also a set of trajectories. That reduces the work of reconstruction. Just simply search the trajectories' id in partitions and the trajectory can be used for distance calculation.

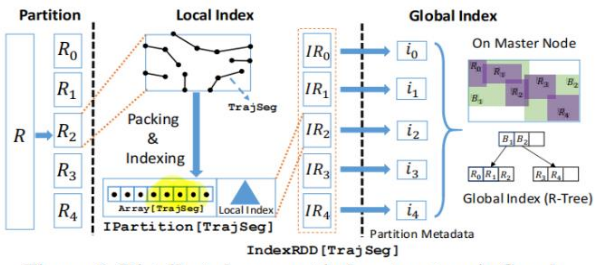


Figure 6: Distributed segment index structure in Spark.

4.3. T3S: Effective Representation Learning for Trajectory Similarity Computation (ICDE, 2021)

T3S [9] is a deep learning-based model to embed trajectory into a vector with low-dimensional space for a given similarity measure. (ie. Trajectories Ti and Tj can be embedded to a vector ti & tj).

The entire geographical space is partitioned into discrete grid cells. Coordinate tuples of a trajectory are mapped to the grid cell. (i.e. Trajectory Tc = [p1, p2, ….., pn], where pi is the coordinate tuple made up of longitude (loni) and latitude (lati), is mapped to sequential representation of grid cell, Tg = [g1, g2, ……, gn], where gi=(xi, yi))

There are 2 main parts in this model. The first part is a self-attention-based network used to capture the structural information. Second part is a LSTM network result to capture the spatial information.

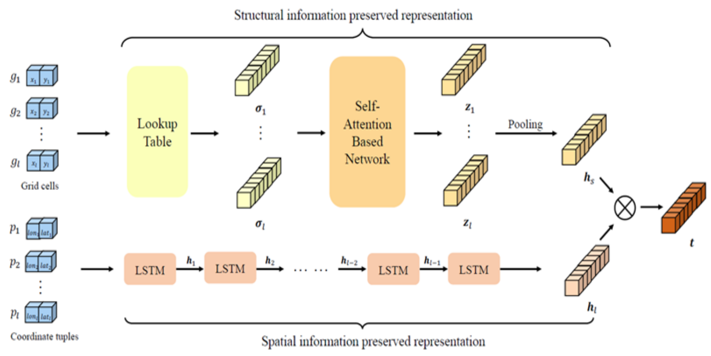

Structural information preserved representation
Grid cells
Lookup Table
Self-Attention Based Network
Pooling
LSTM
Coordinate tuples
Spatial information preserved representation

Fig. Overall structure of T3S

Self-attention-based networks are applied in this model to capture the structural information of trajectories. The input embedding mi is obtained by using the grid representations Tg = [g1, g2, ……, gn] where gi = (xi, yi) and the positional encoding vector. After passing through the network, a weighted value matrix as the representation of grids, zi, is obtained by using softmax as output function. Finally, mean pooling is used to obtain the trajectory representation hs, which provide the structural information.

Given a trajectory T = [(p1, g1); (p2, g2), ……, (pn, gn)], the coordinate tuple pi, where pi = (loni, lati), is feed into LSTM one by one. By the recurrent procedures in LSTM that process the coordinate tuples and capture the correlation among these tuples, the spatial information of the trajectory can be obtained.

Finally, the output of self-attention-based networks (hs) and LSTM (hl) is combined and controlled by using a learnable weight parameter. By adjusting the weight, the output trajectory representation presents both structural and spatial information.

4.4. Discussion

Traditionally, the research may focus on the method to calculate distance and indexing method. In this project, three different solutions have been discussed generally. All of them provide a new direction or a new efficiency improvement method in trajectory similarity computation. Based on a traditional algorithm, “User Oriented Trajectory Similarity Search” proposed a way to provide flexible and highly complex query patterns by introducing a weight to each trajectory query point.

Thanks to advances in technology, distributed data processing systems and machine learning have become more popular, efficient and easy to use. It also impacts the research direction on spatial trajectory similar computation. “Distributed Trajectory Similarity Search” uses spark to implement a distributed trajectory similarity search framework (DFT) which is based on a distributed index structure and some high-performance pruning techniques. Finally, T3S targets on changing the representation of trajectory data. It uses machine learning algorithms to lower the data dimensionality, which can greatly reduce the computation time. Therefore, by combining new technology and traditional algorithms, trajectory similarity search becomes more efficient.

5. Methods for evaluation and experimental results

5.1. User Oriented Trajectory Similarity Search (SIGKDD, 2012)

For user-oriented similarity search, the paper defined only using the R-Tree filtering as MST, and the final algorithm with adaptive grouping filter and refinement as OMST.   The experiment will take the Beijing dataset points as datasets and compare the query performance difference between Naive solution and the optimized solution.

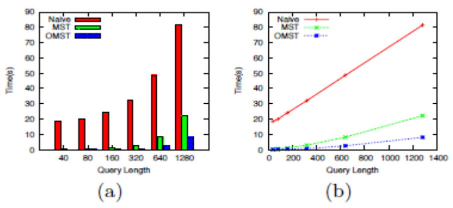


Figure 7: The query time

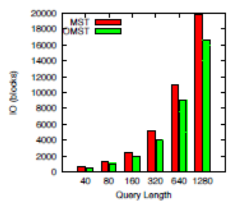


Figure 8: Range query efficiency

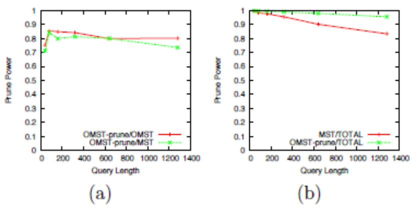
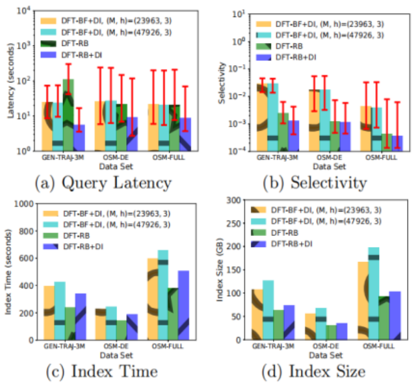


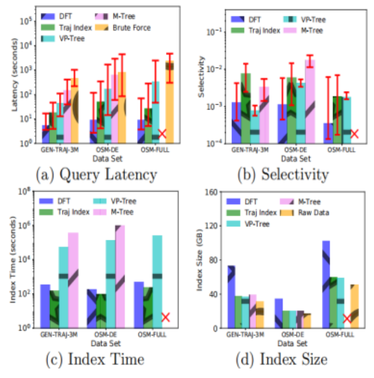
Figure 10: The prune power

5.2. Distributed Trajectory Similarity Search (VLDB, 2017)

As for the experiment objects, it is divided into two parts: DFT internal design and DFT against other solutions. The former generates four different choices to design DFT and apply to test, trying to obtain the best choice. The latter tested DFT against Brute Force, Traj index, VP-Tree, M-Tree, and Centralized.

The metrics of evaluation are query latency, selectivity, index size and index time.





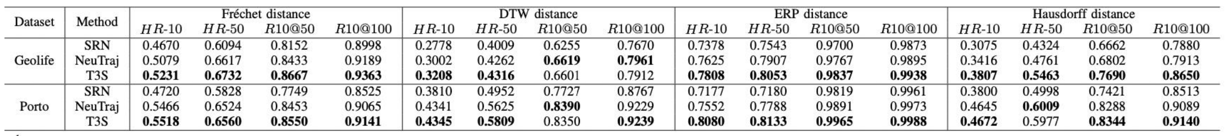
The result shows that as for the internal design, DFT–RB+DI, which is DFT cuts off roaring bitmaps and adopts dual indexing, has led to the best latency and selectivity among all variants. Roaring bitmaps also achieves better index construction time and smaller index size. Dual indexing does not provide much benefit on improving selectivity but helps reduce query latency significantly. Dual indexing does introduce overhead, in terms of both construction time and index size, but this overhead is small and is only a factor of 1.3 to 1.5 compared to without dual indexing. From all the experiments, the design of DFT - RB+DI is comparably the best.

As for the experiments against other baseline solutions, DFT has achieved the smallest query latency. For the selectivity, DFT always beats Traj Index. As for the indexing cost, DFT introduces larger overhead while the other three do not. As for the indexing time, Traj Index performs best followed by DFT.

5.3. T3S: Effective Representation Learning for Trajectory Similarity Computation (ICDE, 2021)

Two real world datasets, Geolife [10] and Porto [11], are used to evaluate the performance of T3S. Geolife is a GPS-based trajectory dataset which contains 17621 trajectories. Porto is a dataset with 1.7 million taxis router trajectories. Also, other machine learning based models, SNR[12] and NEUTRAJ [6], are used to compare with TS3.

PERFORMANCE EVALUATION FOR METHODS UNDER DIFFERENT DISTANCE MEASURES.



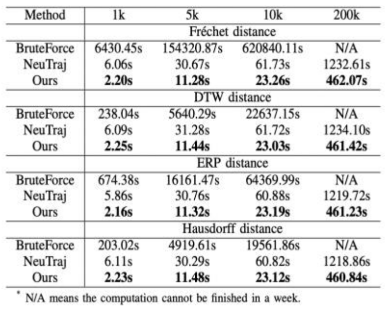
Dataset	Method	Fréchet distance				DTW distance				ERP distance				Hausdorff distance			
		HR-10	HR-50	$R10@50$	$R10@100$	HR-10	HR-50	$R10@50$	$R10@100$	HR-10	HR-50	$R10@50$	$R10@100$	HR-10	HR-50	$R10@50$	$R10@100$
Geolife	SRN	0.4670	0.6094	0.8152	0.8998	0.2778	0.4009	0.6255	0.7670	0.7378	0.7543	0.9700	0.9873	0.3075	0.4324	0.6662	0.7880
	NeuTraj	0.5079	0.6617	0.8433	0.9189	0.3002	0.4262	**0.6619**	**0.7961**	0.7625	0.7907	0.9767	0.9895	0.3416	0.4761	0.6802	0.7913
	T3S	**0.5231**	**0.6732**	**0.8667**	**0.9363**	**0.3208**	**0.4316**	0.6601	0.7912	**0.7808**	**0.8053**	**0.9837**	**0.9938**	**0.3807**	**0.5463**	**0.7690**	**0.8650**
Porto	SRN	0.4720	0.5828	0.7749	0.8525	0.3810	0.4952	0.7727	0.8767	0.7177	0.7180	0.9819	0.9961	0.3800	0.4998	0.7421	0.8513
	NeuTraj	0.5466	0.6524	0.8453	0.9065	0.4341	0.5625	**0.8390**	0.9229	0.7552	0.7788	0.9891	0.9973	0.4645	**0.6009**	0.8288	0.9089
	T3S	**0.5518**	**0.6560**	**0.8550**	**0.9141**	**0.4345**	**0.5809**	0.8350	**0.9239**	**0.8080**	**0.8133**	**0.9965**	**0.9988**	**0.4672**	0.5977	**0.8344**	**0.9140**

[1] HR-k is the hitting ratio for top-k similarity search task.
[2] $R10@k$ is the top-k recall for the top-10 ground truth.

The result shows that T3S is outperforming in both hitting rate and recall on top-k similarity search experiments, especially when using Hausdorff, Fréchet and the ERP distances. Although the local time shift issue is solved when using DTW distance, there still is slight improvement on the hitting rate performance by using T3S.

Porto datasets are randomly sampled with 1K, 5K, 10K and 200K trajectories to evaluate the time cost of similarity computation. Comparing with BruteForce and NeuTraj, T3S provides 100x and 2.5x - 3x faster speed generally. In terms of training time, T3S is also faster than NeuTraj. When using Fréchet or DTW distance, the training time of each epoch is 50% shorter in T3S.

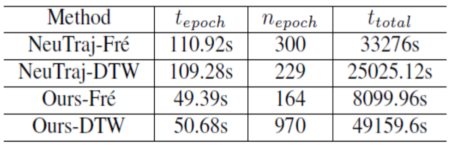
TIME COST OF COMPUTING SIMILARITY UNDER PORTO DATASET WITH DIFFERENT SIZE.



Method	1k	5k	10k	200k
	Fréchet distance			
BruteForce	6430.45s	154320.87s	620840.11s	N/A
NeuTraj	6.06s	30.67s	61.73s	1232.61s
Ours	**2.20s**	**11.28s**	**23.26s**	**462.07s**
	DTW distance			
BruteForce	238.04s	5640.29s	22637.15s	N/A
NeuTraj	6.09s	31.28s	61.72s	1234.10s
Ours	**2.25s**	**11.44s**	**23.03s**	**461.42s**
	ERP distance			
BruteForce	674.38s	16161.47s	64369.99s	N/A
NeuTraj	5.86s	30.76s	60.88s	1219.72s
Ours	**2.16s**	**11.32s**	**23.19s**	**461.23s**
	Hausdorff distance			
BruteForce	203.02s	4919.61s	19561.86s	N/A
NeuTraj	6.11s	30.29s	60.82s	1218.86s
Ours	**2.23s**	**11.48s**	**23.12s**	**460.84s**

* N/A means the computation cannot be finished in a week.

TRAINING TIME FOR DEEP LEARNING BASED METHODS.



Method	t_{epoch}	n_{epoch}	t_{total}
NeuTraj-Fré	110.92s	300	33276s
NeuTraj-DTW	109.28s	229	25025.12s
Ours-Fré	49.39s	164	8099.96s
Ours-DTW	50.68s	970	49159.6s

6. Potential issue

T3S is a model with generality. By adjusting the weight parameter to control the amount of structural and spatial information, it can apply on different dataset and application. However, the model trained out is data oriented, which means the model may not be able to apply on other datasets with different types of trajectory. A specific model needs to be trained on different types of data or application needs. Although the training time is improved in T3S, if the training dataset is large, it is still very time consuming.

7. References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. FODO, pages 69–84, 1993.
[2] E. Keogh. Exact indexing of dynamic time warping. In VLDB, pages 406–417, 2002.
[3] L. Chen, M. ¨Ozsu, and V. Oria. Robust and fastsimilarity search for moving object trajectories. In SIGMOD, pages 491–502. ACM, 2005.
[4] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In ICDE, pages 673–684. IEEE, 2002.
[5] Haitao Yuan and Guoliang Li, "Distributed In-Memory Trajectory Similarity Search and Join on Road Network," 2019 IEEE 35th International Conference on Data Engineering (ICDE), 2019
[6] Di Yao, Gao Cong, Chao Zhang and Jingping Bi, "Computing Trajectory Similarity in Linear Time: A Generic Seed-Guided Neural Metric Learning Approach," 2019 IEEE 35th International Conference on Data Engineering (ICDE), 2019
[7] Wang, H., & Liu, K. "User oriented trajectory similarity search." Proceedings of the ACM SIGKDD International Workshop on Urban Computing - UrbComp, 2012.

[8] Dong Xie, Feifei Li, Jeff M. Phillips, "Distributed Trajectory Similarity Search", VLDB Endowment, Vol. 10, No. 11, pp. 1478-1489, 2017
[9]Peilun Yang, Hanchen Wang, Ying Zhang, Lu Qin, Wenjie Zhangy and Xuemin Liny, "T3S: Effective Representation Learning for Trajectory Similarity Computation," 2021 IEEE 37th International Conference on Data Engineering (ICDE), 2021
[10] Y. Zheng, X. Xie, and W.-Y. Ma, "Geolife: A collaborative social networking service among user, location and trajectory," IEEE Data Eng. Bull., vol. 33, pp. 32–39, 06 2010.
[11] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas, "Predicting taxi–passenger demand using streaming data," IEEE Transactions on Intelligent Transportation Systems, vol. 14, no. 3, pp. 1393–1402, 2013.
[12] W. Pei, D. M. J. Tax, and L. van der Maaten, "Modeling time series similarity with siamese recurrent networks," 2016.