# TRACE-Q: Trajectory Reduction using Accuracy Control for Reliable Querying

Kristoffer M. Gregersen, Mai-Britt L. Laursen, Martin L. Jacobsen,
Nicolai H. Jørgensen, Peter S. Lauridsen, Sarmilan Tharmabalan
Department of Computer Science, Aalborg University, Denmark
{kmgr20, malaur18, mjacob20, njarge20, plauri20, stharm20}@student.aau.dk

***Abstract*—Today's multitude of devices generate a vast amount of GPS data, which is utilized for various applications such as traffic analysis and trajectory studies. However, a significant portion of this data is irrelevant for these applications, resulting in excessive storage usage. Querying this surplus data can be computationally expensive, particularly for applications with limited resources. To address this, researchers have developed algorithms like Bellman, Douglas Peucker, DPhull, and MRPA to reduce dataset size while minimizing the information loss. These algorithms are typically evaluated using error-based metrics, which do not consider the accuracy of queries performed on the compressed data. To overcome this limitation, we propose a query-accuracy driven approach, TRACE-Q, that integrates an efficient simplification algorithm (MRPA) with a query analysis process, thereby preserving query accuracy on the resulting simplified trajectory database. Our experimental results indicate that TRACE-Q surpasses MRPA in terms of both range query accuracy and KNN query accuracy.**

***Index Terms*—Trajectory simplification, trajectory data, query accuracy, benchmarks**

## I. Introduction

In recent years, the ubiquity of object-tracking technologies, such as GPS devices in vehicles and smartphones, fitness- and smartwatches, and computer vision integrated with security and surveillance systems, has led to significant increases in spatio-temporal data in the form of trajectory streams. This surge in data has been leveraged by companies across various domains, who are actively gathering trajectory data for a multitude of purposes. These include travel-route prediction, anomaly detection, behavior pattern mining, and traffic control [1–3].

Large amounts of trajectories are continuously generated. For example, a person equipped with a GPS-enabled device goes shopping while the GPS device records a trajectory point every minute. Shopping for 30 minutes, the person generates 30 trajectory points. While this may not seem significant at first glance, consider a metropolis like Beijing, where approximately 21.5 million inhabitants each possess a GPS-enabled device. Each inhabitant spending an average of 30 minutes shopping, with a sampling rate of one trajectory point per minute would result in roughly 645 million trajectory points in a single day. Over the course of a longer period, this data quickly becomes overwhelming to store and process. To reduce these enormous amounts of data, the original data can be simplified using trajectory simplification algorithms, resulting in a smaller subset of data points [4]. Simplification thus aims to minimize the dataset size while ensuring minimal loss of route information.

Most simplification algorithms use an error-based metric to evaluate the quality of the compressed trajectory [4, 5]. However, this approach ignores the usability of the compressed trajectory i.e. querying. Furthermore, they assume a storage budget that requires a set compression ratio, that can ensure that the storage budget is meet. These types of solutions are called *Error-Driven Trajectory Simplification (EDTS)* [4].

Another branch of simplification algorithms has been proposed by Wang et al. [4] where they use query accuracy in their approach to simplify trajectories. This approach is called *Query accuracy Driven Trajectory Simplification (QDTS)*. A QDTS algorithm preserves the query accuracy as much as possible by considering the trajectory database as one whole rather than simplifying each trajectory one by one. This results in leniency towards the compression ratio of the simplified trajectories, where trajectories are not subjected to a uniform compression ratio [4].

Current EDTS algorithms are limited in their production of usable simplified trajectories, as they consider each trajectory separately during simplification. As pointed out by Wang et al. [4], this may very well be a substandard way of simplifying, as trajectories may be of different complexities and lose valuable information if subjected to the same compression ratio as less complex trajectories. This can ultimately lead to poor query accuracy of the final simplified trajectories.

To address the limitations of poor query accuracy of EDTS algorithms, we extend an EDTS algorithm by applying post-processing query accuracy evaluation to its simplification process. We propose a solution, TRACE-Q, that aims to control the query accuracy while simplifying trajectories, thereby ensuring accurate querying. We use the multiresolution polygonal approximation algorithm MRPA [6] for batch mode simplification, which uses the distance measures integral square synchronous Euclidian distance (ISSD). Our reasoning for choosing MRPA is that Zhang et al. [5] found during experimental evaluation of 25 different simplification algorithms that MRPA consistently outperformed other batch mode algorithms. Incorporating query accuracy into the MRPA algorithm could theoretically improve query accuracy of the simplified trajectories and make the simplification more valu-

able. The query accuracy will drive the process and influence the compression ratio, implying that the trajectory simplification does not operate within a specific storage budget. While the definition of the QDTS problem states that the trajectory database is to be considered as one whole [4], our approach considers this definition too narrow. In our approach, we show that a simplification algorithm can consider each trajectory separately and still be driven by query accuracy, thus expanding the definition of the QDTS problem by removing the aforementioned limitation. Through this approach, we aim to achieve more efficient and suitable simplifications that will be more valuable for querying, thereby addressing the current limitations in the field. This research will contribute to the development of more effective data simplification algorithms and enhance the efficiency of data querying processes.

To summarize, we make the following contributions:

- We propose a QDTS algorithm called TRACE-Q that utilizes an EDTS algorithm to achieve greater usability of the simplified trajectories, while aiming to strike a balance between compression and query accuracy.
- We utilize the unique multiresolution polygonial approximation of the MRPA algorithm alongside query accuracy to simplify trajectories to achieve greater query accuracy compared to a solely error-driven algorithm.
- We make our implementation details public through our GitHub[1].
- We conduct experiments using trajectory data from the T-Drive dataset [7, 8]. Our experiments show TRACE-Q consistently outperforming MRPA by achieving a higher $F_1$ score in both range and KNN query accuracy experiments.

The rest of this paper is structured as such: Section II present relevant theory on the subject of trajectories, simplification, the methods and technologies used in our approach, and our problem definition. Section III present related works in the field of batch mode simplification. Section IV explains our methodology. Section V presents the results of our experiments. Lastly, Section VI concludes on this approach to trajectory simplification.

## II. Preliminaries

This section covers the fundamentals of trajectory theory, going over trajectories, simplification theory, and evaluation of simplification methods. Furthermore, we describe the MRPA algorithm, how the range and W-KNN queries are defined, and how the quality is measured. Lastly, the problem is defined.

### A. Trajectory

A trajectory $\tau$ denotes a sequence of time-stamped points that describe the movement of an object over time. We denote the number of points in a trajectory as $|\tau|$. The notation of a trajectory point is a tuple $(x_i, y_i, t_i)$, where $p = (x_i, y_i)$ is the location information and $(t_i)$ is the recorded timestamp of the given point. The location is noted as longitude and latitude. A trajectory is noted as a series of points $p_1, p_2, ..., p_n$.

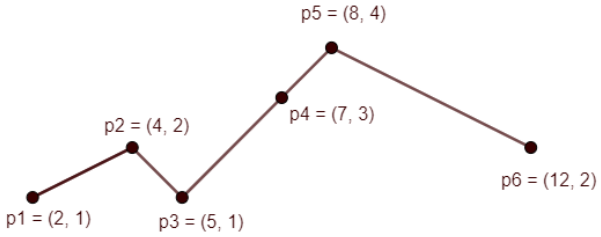


Fig. 1: An example trajectory.

Figure 1 shows an example trajectory, consisting of six points $p_1, p_2, p_3, p_4, p_5, p_6$. Each point is connected by a line segment denoted as $\overrightarrow{p_i p_{i+1}}$. Many trajectory data use cases make use of the direction in which the object travels along the trajectory. Each line segment in a trajectory also has its own direction, denoted $\theta(\overrightarrow{p_i p_{i+1}})$.

### B. Trajectory Simplification

Trajectory simplification is the process of reducing the number of points in a trajectory $\tau$ to produce a new simplified trajectory $\tau_s$ such that $|\tau| \geq |\tau_s|$.

There are two problems considered in trajectory simplification. The *min-#* problem refers to the goal of minimizing the number of points retained while guaranteeing that the simplification error $\varepsilon_{simp}$ is bounded by a given value. On the other hand, the *min-*$\varepsilon$ problem is concerned with finding a simplification with a given number of points that minimizes the error measure $\varepsilon_{simp}$.

Simplification algorithms are divided into two categories: **Online mode** and **Batch mode**. Online mode considers continuous simplification, where a trajectory is given to the simplification system one point at a time. This category of simplification is typically used in real-time systems with continuous sampling of points. Given the nature of online simplification, the simplification system does not know the entire trajectory to be simplified, and thus is unable to achieve optimal results in the *min-#* and *min-*$\varepsilon$ problems.

Batch mode simplification algorithms, as opposed to online mode, are given a complete trajectory to be simplified. When the simplification system knows the entire trajectory, it can achieve optimal *min-#* and *min-*$\varepsilon$ solutions.

### C. Error-Based Quality evaluation

Error-based quality metrics aim to measure the $\varepsilon_{simp}$ of a simplification algorithm, done typically by some distance measuring approach. Figure 2 visualises three of the most common error-based metrics, Perpendicular Euclidean Distance (PED), Synchronized Euclidean Distance (SED), Direction-Aware Distance (DAD). These metrics use Euclidean and angular distance, while position, speed, and orientation have also been considered as error measures in literature [6].

[1] https://github.com/KarmaKamikaze/TRACE-Q

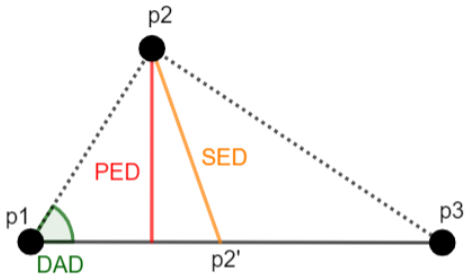


Fig. 2: The error-based metrics, PED, SED and DAD.

**PED** is the Euclidean distance from a point perpendicular to a line. This is the shortest distance from the point to the line. In Figure 2 we see the point $p_2$ is perpendicular on $\overrightarrow{p_1p_3}$.

**SED** is the Euclidean distance between a point and its synchronized point on the line segment, where the synchronized point is estimated via interpolation between its predecessor and successor. This is exemplified by Figure 2 where the point $p_2'$ on $\overrightarrow{p_1p_3}$ is the synchronized point of $p_2$.

**DAD** is the difference in direction between $\overrightarrow{p_1p_2}$ and $\overrightarrow{p_1p_3}$ measured in angular distance. In Figure 2 the angular difference is measured between $\overrightarrow{p_1p_2}$ and $\overrightarrow{p_1p_3}$.

**ISSD** is an extended version of Integral Square Error (ISE), where ISE sums the calculated square of $\varepsilon_{simp}$; this is useful in approximating polygonal curves [6]. In Figure 3 $\varepsilon_{simp}$ is estimated as $d_1'^2 + d_2'^2 + d_3'^3$ from the approximated curve $\overrightarrow{p_1p_4p_6}$ where $d'$ is the synchronized point.

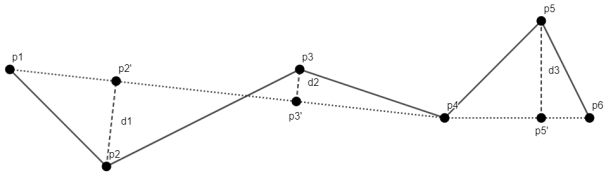


Fig. 3: ISSD is estimated as $d_1'^2 + d_2'^2 + d_3'^3$..

### D. Query-Based Quality evaluation

Trajectories are typically stored in a database where they support querying. While error-based metrics are good at assuring optimal solutions to the *min-#* and *min-*$\varepsilon$ problems, they neglect the usability of the resulting simplified trajectories. Query-based quality measures instead measures quality in terms of usability. This is done by performing spatio-temporal queries on the trajectory database and measuring their accuracy.

Zhang et al. [5] performed query-based quality evaluation on several trajectory simplification algorithms, using three popular spatio-temporal queries, namely *Window Range Query*, *Window KNN Query*, and *Window Trajectory Distance Join*. These queries are commonly used to measure the query accuracy of simplified trajectories.

### E. MRPA algorithm

In this section, we provide a concise overview and explanation of the MRPA algorithm for GPS trajectory simplification proposed by Chen et al. [6]. The MRPA algorithm is a fast polygonal approximation algorithm in 2-D space that operates under the integral square synchronous distance error criterion. It achieves this in linear time complexity, making it highly efficient for large datasets. The algorithm consists of three sequential procedures.

*1) Error tolerance initialization:* The algorithm initialises $\log_c N$ error tolerances $e_1^*, e_2^*, ...$ The tolerances are estimated according to the integral square synchronous Euclidean distance (ISSD) and local integral square synchronous Euclidean distance (LSSD) criterions as:

$$e_k^* = \frac{1}{N/c^k - 1} \sum_{1 \le j < N/c^k} \delta(P_{i_j}^{i_{j+1}}) \tag{1}$$

$$i_j = \frac{N-1}{N/c^k - 1} \cdot (j-1) + 1. \tag{2}$$

This estimation is the average error for ISSD/LSSD for all approximated segments when the curve is equally partitioned.

*2) Initial curve approximation:* The initialized error tolerances are used in a bottom-up multiresolution algorithm to estimate the approximated curves $P_1^*, P_2^*, ..,$ with decreasing resolution. Here, resolution refers to the size of the approximation. At each approximation result, the resolution is affected by the used error tolerance, meaning the resolution lessens at each estimation. The result of the previous approximation is used in the subsequent approximation.

*3) Final approximation:* Given error tolerance $\epsilon$, the final approximation is found by selecting the best-suited curve $P_k^*$ from the results in initial curve approximation by:

$$k = \texttt{arg}\,\texttt{max}_k(e_k^* < \epsilon) \tag{3}$$

Chen et al. [6] found that the MRPA algorithm performed on-par or outperformed other state-of-the-art algorithms in terms of processing time and error performance. This corresponds to the findings of Zhang et al. [5], when comparing the performance of 25 trajectory simplification algorithms. They concluded that the MRPA algorithm is the preferred choice, when the primary focus is simplification and query accuracy.

### F. Query

In this section, we discuss our choices of spatio-temporal queries used in our approach. As previously mentioned in Section II-D, [5] employed three distinct queries to test and evaluate various simplification algorithms. Their chosen spatio-temporal queries are, as they also stated, commonly used. Based on this and their commonality, we have decided to evaluate our approach using range and W-KNN queries. We define the two queries as follows:

*1) Range query:* Given a database of trajectories and a range window $(q_{x_{min}}, q_{x_{max}}, q_{y_{min}}, q_{y_{max}}, q_{t_{min}}, q_{t_{max}})$, a range query finds all trajectories in the database that are partly contained within the range window, meaning that at least one point $p_i = (x_i, y_i, t_i)$ in a trajectory must satisfy $q_{x_{min}} \le$

$x_i \leq q_{x_{max}}$, $q_{y_{min}} \leq y_i \leq q_{y_{max}}$, and $q_{t_{min}} \leq t_i \leq q_{t_{max}}$. Figure 4 shows a simple range query. The original trajectory is colored blue and the simplified trajectory is colored red. Each trajectory point is timestamped, starting at timestamp 0. The range window is colored orange with time window $[2, 4]$. In this example, only the original trajectory is in the window, and therefore the result of the range query would only contain the original trajectory.

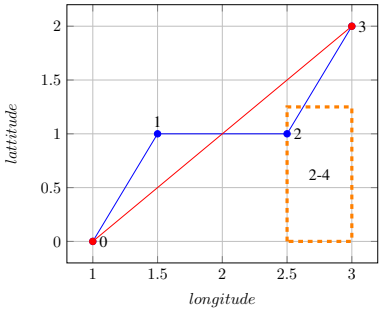


Fig. 4: Illustration of a range query. Blue is the original trajectory, red is the simplified trajectory, and orange is the range window.

*2) W-KNN query:* Given a database of trajectories, an origin point $p = (x, y, t_{min}, t_{max})$, and a value $k$, a W-KNN query finds the $k$ nearest trajectories to $p$ that are also contained within the time window provided by $p$. We use the Euclidean distance formula to calculate the distance between the origin point and another trajectory point. Figure 5 shows simple W-KNN query. The blue, red, and green trajectories are original trajectories and purple is a simplified variation of the green trajectory. Each trajectory point is timestamped, starting at timestamp 0. The orange point is the origin point with the time window $[0, 2]$. In this example, we have that $k = 3$. Here, the simplification process did not affect the query result. For the remainder of the paper, we abbreviate W-KNN to KNN.

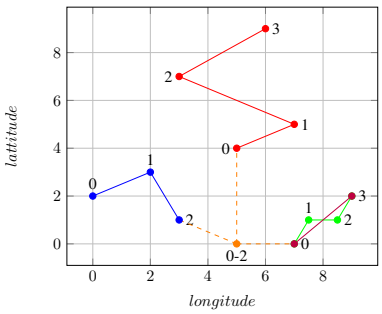


Fig. 5: Illustration of a W-kNN query. blue, red, and green are original trajectories, and purple is the simplified trajectory. The orange point is the origin point.

### G. Quality measures

To measure the difference between the original and the simplified trajectories we will use the $F_1$-score. The original trajectories are used as the ground truth and then the quality of the results from the simplified trajectories are measured using the $F_1$-score. When the $F_1$-score is large it indicates that there is a smaller difference between the query results. The $F_1$-score is defined by

$$F_1 = \frac{TP}{TP + 1/2(FP + FN)} \tag{4}$$

where True Positives (TP) are the intersection between the original trajectories and the simplified trajectories. The False Positives (FP) are the simplified trajectories minus TP, while False Negatives (FN) are the original trajectories minus TP.

### H. Problem Definition

We propose an extension to the QDTS problem by considering each trajectory to be simplified separately, but still having a leniency towards compression ratio. This is done by utilizing the MRPA algorithm and applying post-processing query accuracy evaluation to its simplification result.

The problem is then: Given a trajectory database $T_D$ and a maximum query error $\varepsilon_{query}$, TRACE-Q aims to produce a trajectory database $T'_D$ of simplified trajectories, such that querying $T'_D$ provides results that differ no more than $\varepsilon_{query}$ compared to the same queries on $T_D$.

## III. Related Work

In this section, we cover other batch mode simplification algorithms. Based on the findings in [5], we only cover position-preserving algorithms, given that direction-preserving algorithms have been found to work poorly in practice.

### A. Position-Preserving and Direction-Preserving

Many trajectory simplification algorithms are *position-preserving*, meaning they are concerned with simplifying trajectories such that the positional information is captured in the simplified trajectory. A *direction-preserving* algorithm preserves the direction of a trajectory segment, by considering the angle of the anticlockwise rotation from the positive x-axis [9]. Direction-preserving algorithms thus aim to preserve critical information, that may be lost in position-preserving algorithms. However, while direction-preserving algorithms are theoretically better at preserving valuable trajectory information, they appear to work poorly in practice. Zhang et al. [5] has performed quality analysis of 25 different simplification methods on 5 GPS datasets, where they noted that these types of algorithms are only recommended in applications there the direction of a trajectory is the most important aspect.

### B. Classical Trajectory Simplification Algorithms

The first trajectory simplification was proposed by Richard Bellman in 1961 and used dynamic programming to find an approximated trajectory that has the minimum spatial distance error (*min-*$\varepsilon$), all achieved with a time complexity of $\mathcal{O}(N^3)$ [10].

David H Douglas and Thomas K Peucker proposed the DP algorithm in 1973 as an approximation method to simplification [11]. The algorithm takes a trajectory segment and an error threshold $\epsilon$ and finds a point $p$ within the segment that causes the greatest perpendicular distance. If that greatest distance is

less than the given $\epsilon$, the points between the two end points are discarded reducing segment to only its two end points. Otherwise, if the perpendicular distance is greater than $\epsilon$, the trajectory segment is split on $p$ into two smaller trajectories. Then, the procedure is run recursively on the trajectories. The DP algorithm achieves a time complexity $\mathcal{O}(N^2)$.

In 1992 John Hershberger and Jack Snoeyink proposed an update to the DP algorithm named DPhull [12], incorporating convex hull to avoid the process of finding the point $p$ with the greatest perpendicular distance. This reduced the time complexity to $\mathcal{O}(N \log N)$ while generating the same results as the original DP algorithm.

In 2004, another variation of the DP algorithm was proposed by Nirvana Meratnia and Rolf A. de By named TD-TR (top-down time-ratio) [13]. It is very similar to the DP algorithm, with the most notable difference being the distance measure. TD-TR uses the SED distance measure taking advantage of the temporal dimension, as opposed to DP which used the perpendicular distance measure. TD-TR achieves the same time complexity of $\mathcal{O}(N^2)$ as the DP algorithm.

### C. Query Accuracy Driven Trajectory Simplification

The work by Wang et al. [4] introduces an approach that collectively simplifies multiple trajectories in a database where the focus is on query accuracy. This approach considers the interrelationship between the trajectories to optimize the simplification process while ensuring that the resultant dataset maintains high usability for spatial queries. They suggest the RL4QDTS algorithm that utilizes multi-agent reinforcement learning. Their experiments show that RL4QDTS outperforms existing EDTS algorithms on real-world trajectory datasets regarding query accuracy.

## IV. Methodology

This paper proposes a new algorithm called TRACE-Q[2] for QDTS. It simplifies each trajectory separately and utilizes an EDTS algorithm to create a set of simplifications with decreasing size. Numerous range and KNN queries are then performed on each simplification to identify the optimal compression that achieves an acceptable $F_1$-score, which is then selected as the fitting simplification for the given trajectory. We have selected MRPA as the EDTS algorithm, since its produced simplifications preserve query accuracy more than other EDTS algorithms, as described in Section II-E. Furthermore, in the first steps of MRPA, a set of simplifications with decreasing size is created, followed by a last step, wherein a final approximation is determined based on a given error tolerance. By omitting this last step of MRPA, we can directly use the remaining part of MRPA in TRACE-Q. Other EDTS algorithms do not produce such a set, and therefore one would need to execute those algorithms with varying error tolerances to produce a set of simplifications. We present the details of TRACE-Q in the following sections. Section IV-A describes the framework for simplifying a single trajectory, while Section IV-B describes how the range and KNN queries used to determine the $F_1$-score are constructed. Lastly, Section IV-C describes how the $F_1$-score for a simplification is calculated.

[2]https://github.com/KarmaKamikaze/TRACE-Q

### A. TRACE-Q Simplification Framework

To simplify each trajectory separately, we utilize Algorithm 1. The *Simplify* algorithm takes as input the trajectory to be simplified along with two minimum $F_1$-scores, one for range queries and one for KNN queries. These are used to determine whether a specific simplification preserves an acceptable query usability. The algorithm starts by constructing two separate sets of objects we call *Query tests*. These objects contain all the necessary information needed to perform either a range query or a KNN query. In the case of range queries, they contain a spatiotemporal window, while the KNN query tests contain a spatial origin point with an associated time window. Additionally, when constructing these query tests, the query is also performed on the original trajectory, and the result is stored in the object. Range query tests are performed only on the original trajectory, and therefore store the result as a boolean value that determines whether the original trajectory is inside the window. KNN query tests however are performed on the whole original dataset, as KNN queries consider the relation between trajectories. Therefore, the query result is stored as a list of trajectories describing the k nearest neighbors.

Utilizing these query tests enables uniform query usability testing on multiple simplification variants of the same original trajectory while avoiding unnecessarily performing the query on the original trajectory repeatedly. The creation of these query tests will be further described in Section IV-B. The *Simplify* algorithm then utilizes MRPA without performing the last step of final approximation to create a set of simplifications with decreasing resolution. This list is reversely iterated through to find the smallest simplification that has range query and KNN query $F_1$-scores that are greater than the user-provided minimal. If no simplification has acceptable $F_1$-scores, the original trajectory is instead returned. Thereby, we prioritize maintaining query usability rather than achieving a high compression ratio.

**Algorithm 1** Simplify(t, rq_mf1, knn_mf1)

$rqts \leftarrow CreateQueryTests(t, rq\_gd, rq\_tim, rq)$
$kqts \leftarrow CreateQueryTests(t, knn\_gd, knn\_tim, knn)$
$sts \leftarrow MRPA(t)$
**for** $st$ in $sts.reverse$ **do**
  $[rq\_f1, knn\_f1] \leftarrow CalculateF1(t, st, rqts, kqts)$
  **if** $rq\_f1 \geq rq\_mf1$ and $knn\_f1 \geq knn\_mf1$ **then**
    **return** $st$
  **end if**
**end for**
**return** $t$

### B. Query Tests

To construct the range and KNN query tests, we utilize the process outlined in Algorithm 2 called *CreateQueryTests*. In essence, this algorithm constructs a spatiotemporal grid over

the original trajectory, where several query tests are created for each point in the grid. The algorithm takes as input the original trajectory and two parameters used for constructing the grid called *gd* and *tim*. *gd* stands for *Grid Density*, and it defines how spatially close points in the grid should be. Therefore, lower values of *gd* allow for more points in the grid. This, in turn, means that more unique query tests can be constructed, thereby making the $F_1$-score calculation more precise at the cost of computation time. *tim* stands for *Time Interval Multiplier*, and it defines how temporally close points in the grid should be, and it has the same properties as *gd*. Separate *gd* and *tim* values can be used for range and KNN query tests. The last parameter is an enumerator that defines whether range query tests or KNN query tests should be constructed.

The algorithm first determines the MBR of the trajectory, whereafter it slightly expands the spatial coordinates of it based on *gd*. As the grid will be constructed over this MBR, expanding it ensures that edge-case query tests are created. The *gd* and *tim* parameters are then used to determine how many points there should be on the spatial and temporal axes of the grid. The algorithm then iteratively constructs the grid points based on the expanded MBR and the *gd* and *tim* values. For each grid point, the aforementioned enumerator is used to determine whether range query tests or KNN query tests should be created.

In the case of range query tests, several range queries with increasing window size are created for each grid point. As a baseline, the smallest window created has a size corresponding to the distance to the next grid point, which ensures that all of the MBR's area is considered in the query tests constructed. The amount of windows created for a grid point and the factor by which each window increases in size is user-defined. This factor is called *Window Expansion Rate*. A greater amount of windows cause the $F_1$-score to be more precise since more range query tests are created, however, it comes at a cost of computation time.

In the case of KNN query tests, a single query is created for each spatiotemporal point in the grid. Furthermore, a query is also created for each spatial point in the grid, where the temporal region is set to the minimum and maximum possible values. This ensures that we consider KNN queries that do not have a time window. The *k* value used to determine how many neighbors should be analyzed is user-defined. Smaller *k* values cause the result list to be smaller, thereby reducing the possibility that the simplification would be in the result list. Therefore, smaller *k* values cause the $F_1$-score to be more precise.

### C. Calculating $F_1$-scores

To calculate the total $F_1$-scores of the range query tests and KNN query tests for an original trajectory and its simplification we utilize Equation (4). Note that while the $F_1$-score is based on the false positive cases, we do not consider these in TRACE-Q. This is because the locations of the simplifications are a subset of the original trajectory, which means that false positives are not possible for the queries. For range queries, the simplification can't be in the window if the original trajectory is not, since this would imply that the simplification contains a location not present in the original trajectory. KNN queries can have false positives when the queries are performed on the complete set of original trajectories, whereafter they are performed on the complete set of simplified trajectories. However, when evaluating the simplification on the KNN query tests described in Section IV-B, the original trajectory is replaced with its simplification while keeping the other trajectories constant. In this case, we can see how false positives are impossible as they would imply that the simplification has a location that is closer to the query origin than that of any location in the original trajectory.

**Algorithm 2** CreateQueryTests(t, gd, tim, rq_or_knn)

```
qts ← {}
mbr ← CalculateAndExpandMBR(t, gd)
coordinate_points ← 1/gd
time_points ← 1/tim
for x = 0 to coordinate_points do
    x_coord ← mbr.x_low + x*gd*(mbr.x_high − mbr.x_low)
    for y = 0 to coordinate_points do
        y_coord ← mbr.y_low + y * gd * (mbr.y_high − mbr.y_low)
        for ti = 0 to time_points do
            ti_value ← mbr.ti_low + ti * tim * (mbr.ti_high − mbr.ti_low)
            if rq_or_knn == rq then
                qts.insert(CreateRangeQueryTests(t, x_coord, y_coord, ti_value, mbr))
            else
                qts.insert(CreateKNNQueryTests(t, x_coord, y_coord, ti_value, mbr))
            end if
        end for
    end for
end for
return qts
```

## V. Experiments

To conduct our experiments we use the T-Drive dataset. The T-Drive dataset contains 10.357 GPS trajectories of taxis in Beijing spanning over a period of one week. The total number of points is about 15 million, which is a reasonable amount of data to perform our experiments on.

We will utilize two performance metrics in our experiments: $F_1$-score and Compression Ratio. In our experiments, we will determine the $F_1$-score metric by performing a large number of queries in a grid-like fashion, closely following the approach outlined in Section IV-B. The Compression Ratio indicates the level of simplification the algorithm achieves. It defines the factor by which the simplified data has been compressed. For example, a compression ratio of 10 means that the simplified data becomes 10 times smaller than the original data.

The baseline we will compare to is our implementation of MRPA. By comparing it against MRPA, we can analyze how well our approach improves upon it regarding the preservation of query accuracy.

Our experiments were conducted on an Intel i7-9750H CPU with 16 GB RAM running Windows 11 Home.

The code for our implementation of TRACE-Q can be found on our GitHub[3].

### A. Preliminary TRACE-Q Parameter Tweaking

We conduct a series of experiments using a random sample of 200 trajectories from the T-Drive [7, 8] dataset to evaluate TRACE-Q's effectiveness and efficiency. The parameters under test and the experimental results are shown in Figure 6.
*1) Comparison of Compression Performance With and Without KNN Queries:* The first set of experiments focuses on assessing the runtime performance of TRACE-Q. We compare the performance of our algorithm when run with and without the utilization of KNN query test objects, as Figure 6-a depicts. Alongside the runtime performance, we analyze the simplified trajectories' compression ratio, as shown in Figure 6-b. This is again done with and without the use of KNN query tests. Additionally, it is important to note that each experiment is calibrated using identical $F_1$-scores.

The results of these experiments reveal significant insights. When KNN queries are incorporated as part of the simplification process, we observe a considerable increase in the compression ratio. This suggests that the inclusion of KNN queries enhances the simplification process by yielding more compressed trajectories. However, it's important to note that the use of KNN queries also results in a significant increase in runtime. This finding underscores the trade-off between compression efficiency and runtime performance, a crucial consideration for algorithm developers and data scientists. Despite this, we believe that the trade-off is acceptable for our purposes, as our primary focus is on achieving a higher compression ratio.
*2) Impact of K-Values in KNN Queries:* We conduct additional experiments to investigate the impact of different *k*-values in KNN queries on the algorithm's performance. Specifically, we examine the correlation between the Compression Ratio and k-value, as well as the $F_1$-score per *k*-value.

Our findings indicate that the Compression Ratio rises steeply for lower values of *k*, as seen in Figure 6-c. However, this increase quickly subsides for higher values of *k*. This suggests that considering all trajectories (i.e., when $k = N$) might intuitively yield better compression, but the actual benefits diminish rapidly beyond a certain *k*-value. On the other hand, the $F_1$-score of KNN query in Figure 6-d shows a consistent decrease as *k* increases. This is because each KNN query test becomes easier to pass as more trajectories are considered for each test. This trade-off between Compression Ratio and $F_1$-score underscores the need for a balanced approach in selecting the *k*-value.

Considering these findings, although the results can be argued to be uncertain because of the low amount of data, we recommend a *k*-value that is not too high. It should only encompass the immediate proximity of the origin point. This approach offers a boost to the Compression Ratio without suffering from diminishing returns. It also helps maintain a reasonable $F_1$-score, ensuring the accuracy of the KNN queries.
*3) Impact of Range Query Grid Density and Time Interval:* We conducted experiments to examine the impact of *Range Query Grid Density* and *Range Query Time Interval* on the algorithm's performance. These experiments were designed to observe their effects on the Compression Ratio and the $F_1$-score for both range queries and KNN queries.

Figure 6-e shows that the Compression Ratio increases with higher grid density and time interval. However, the curve shows significant diminishing returns beyond a certain grid density and time interval value. This indicates that the Compression Ratio is not significantly affected by extremely high values of grid density and time interval. Conversely, the $F_1$-scores for both range queries and KNN queries decrease in an exponential manner as grid density and time interval values increase. This is illustrated in Figure 6-f. The decrease in $F_1$-scores is because the query tests become easier to pass with higher grid density and time interval values since higher values result in larger windows as described in Section IV-B.

These findings highlight the importance of carefully selecting the grid density and time interval values when using the TRACE-Q algorithm. While higher values can improve the Compression Ratio, they also make the query tests easier to pass, thereby reducing the $F_1$-scores. Therefore, finding a balance that maximizes the Compression Ratio without excessively compromising the $F_1$-scores is crucial.
*4) Impact of KNN Query Grid Density and Time Interval:* We similarly conduct experiments to examine the impact of *KNN Query grid density* and *KNN Query time interval* on the algorithm's performance. These experiments are designed to observe their effects on the Compression Ratio and the $F_1$-score for both range queries and KNN queries.

As shown in Figure 6-g, the Compression Ratio decreases with higher KNN grid density and time interval values. This is an aftereffect of how KNN queries are incorporated into the simplification process of TRACE-Q. We perform an extra KNN query per spatial grid point that considers the entire time interval, as described in Section IV-B. This query test is naturally more challenging to pass than tests with a shorter time interval slice because all other trajectories' points may be considered. Therefore, with higher values of KNN grid density and time intervals, fewer query tests contain specific time slices, making it so that the extra full-time-sliced query tests constitute a more significant amount of the total query tests, contributing to a more considerable impact on the Compression Ratio. On the other hand, the $F_1$-score for range queries is not heavily dependent on the values for KNN grid density and time interval. However, the $F_1$-score for KNN is heavily influenced by higher values of grid density and time interval. This is illustrated in Figure 6-h.

The relationship between Compression Ratio and $F_1$-score is, in this case, contradictory to the negatively correlated pattern shown for *Range Query Grid Density* and *Range Query Time*

[3]https://github.com/KarmaKamikaze/TRACE-Q

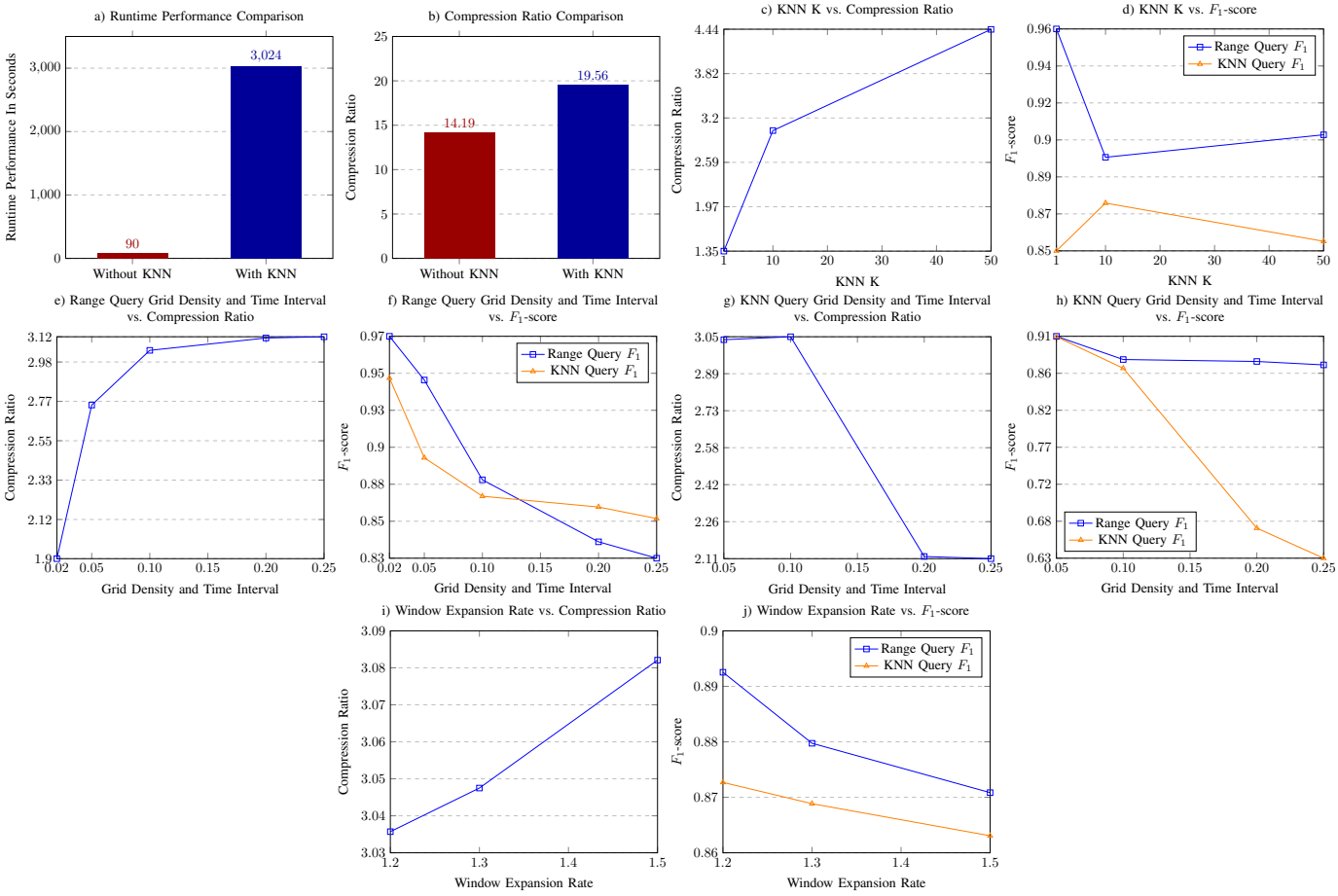


Fig. 6: TRACE-Q Experiments. *a* and *b* were measured using the same $F_1$-score in both experiments and show the runtime performance and compression ratio differences when incorporating KNN queries in the simplification process. The range query $F_1$-score was `0.8434` while the KNN query $F_1$-score was `0.6728`. *c* and *d* show the compression ratio and $F_1$-score for varying KNN k values. *e* and *f* show the compression ratio and $F_1$-score for varying grid density and time intervals for range queries. *g* and *h* show the compression ratio and $F_1$-score for varying grid density and time intervals for KNN queries. *i* and *j* show the compression ratio and $F_1$-score for varying window expansion rates.

*Interval* in the previous section. Intuitively, one would assume that a lower Compression Ratio would result in a higher $F_1$-score; however, here, the Compression Ratio and $F_1$-score are positively correlated. We suspect that this is due to the extra KNN query test with no temporal region that we perform for each spatial grid point. While this query does cause the Compression Ratio to be lower for high values of grid density and time interval, the simplifications that TRACE-Q produces when this query has a high impact may not be representative of the key qualities of the original trajectory, and therefore, the $F_1$-scores for high values of grid density and time interval are low despite also having low Compression Ratios.

These findings have significant implications for the practical application of the TRACE-Q algorithm. They underscore the need for careful selection of the KNN grid density and time interval values. High values can have a detrimental effect on the Compression Ratio and the KNN $F_1$-score, suggesting that lower values may be more suitable. Further experiments on even lower values of *KNN Query grid density* and *KNN Query time interval* could provide a clearer understanding of the lower bounds of suitable values.

*5) Impact of Window Expansion Rate:* The final set of preliminary experiments we perform focuses on observing the impact varying *Window Expansion Rate* values have on Compression Ratio and $F_1$-score. The *Window Expansion Rate*, as described in Section IV-B, is the degree to which windows at each grid point expand.

The results of these experiments, as seen in Figure 6-i and Figure 6-j, reveal interesting insights. Higher values of *Window Expansion Rate* scale up the windows more quickly, making the resultant range of queries easier to pass because of the larger windows. This leads to a near-linear relationship in the results, where the Compression Ratio and $F_1$-scores are negatively correlated. However, these findings also indicate that we cannot suggest a value of *Window Expansion Rate* that is universally well-suited for all scenarios. The optimal value may vary depending on the application's specific requirements, such as the desired balance between Compression Ratio and $F_1$-score.

### B. MRPA Baseline vs. TRACE-Q

We present an experimental study that compares our proposed TRACE-Q algorithm with the established MRPA baseline. Our evaluation bases on a random sample of 500 trajectories for each experiment, extracted from the T-Drive dataset, each time using a fresh set of data, ensuring a thorough dataset coverage. We carefully select the parameters for both algorithms to ensure that the results fall within a similar compression ratio range. Those parameters, shown in Table I, are based on the preliminary parameter tweaking performed in Section V-A.

| TRACE-Q Parameters | |
|---|---|
| **Parameter** | **Value** |
| KNN-$k$ | 10 |
| Range query grid density | 0.1 |
| KNN query grid density | 0.1 |
| Range query time interval | 0.1 |
| KNN query time interval | 0.1 |
| Window expansion rate | 1.3 |

TABLE I: The TRACE-Q Parameters chosen based on the preliminary tweaking.

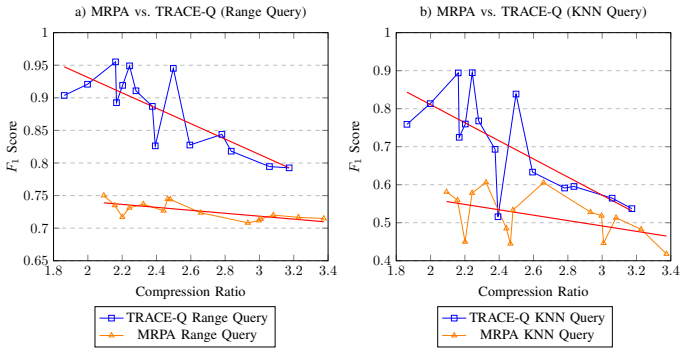


Fig. 7: MRPA baseline vs. TRACE-Q Experiments. Plot *a* shows the correlation between compression ratio and $F_1$-score related to range query accuracies on experiment data for the MRPA baseline and TRACE-Q, respectively. Plot *b* shows the correlation between compression ratio and $F_1$-score related to KNN query accuracies.

Our primary comparison metric is the $F_1$-score for a specific compression ratio. We evaluate and compare the performance of both algorithms in terms of range query accuracies and KNN query accuracies. The results of these experiments are visually represented in Figure 7. This section discusses the key findings and implications of the results shown in these figures.

Figure 7-a illustrates the $F_1$-scores associated with range query accuracies for both MRPA and TRACE-Q. Notably, TRACE-Q consistently demonstrates higher query accuracy compared to MRPA across the tested compression ratios. This is visually represented by the linear regression lines, with TRACE-Q's line situated above that of MRPA, indicating superior performance in terms of $F_1$-score. This suggests that the simplifications generated by TRACE-Q are more effective for subsequent range querying tasks, at least within the range of compression ratios evaluated in this study. Figure 7-b shows a similar plot but focuses on the $F_1$-scores for KNN query accuracies. Here too, TRACE-Q outperforms MRPA in KNN query accuracy, although the data set appears more fluctuating than for range queries. This indicates that KNN queries are more affected as the compression ratio increases. Nonetheless, TRACE-Q's linear regression line appears significantly higher, showing better $F_1$-scores for all tested compression ratios, indicating yet another advantage for TRACE-Q.

Several critical observations can be drawn from these figures:

1) **Superior Query Accuracy of TRACE-Q:** TRACE-Q maintains a higher $F_1$-score across all tested compression ratios for both range and KNN queries, underscoring its effectiveness in preserving essential trajectory information. This consistent superiority implies that TRACE-Q produces simplifications that retain more meaningful data compared to MRPA.
2) **Potential Intersection Point:** The plots indicate a tendency for the $F_1$-scores of TRACE-Q and MRPA to converge at very high compression ratios. This warrants further investigation through additional experiments at higher compression ratios to determine if and when the $F_1$-scores of the two algorithms intersect. Such an intersection would provide insights into the limits of TRACE-Q's superiority and the compression thresholds at which MRPA might become more competitive. However, the $F_1$-score for TRACE-Q may prove to be non-linear, in which case it will likely always yield better results.
3) **Stability of $F_1$-Score:** The $F_1$-score for TRACE-Q appears to fluctuate less and stabilize at higher compression ratios, especially for range queries. This suggests that the algorithm reaches a point where further compression does not significantly degrade query accuracy. In contrast, MRPA's $F_1$-score remains relatively stable across different compression ratios, showing no significant improvement or degradation. For KNN queries, both algorithms show more instability, but TRACE-Q again stabilizes slightly at higher compression ratios.
4) **Implications for Practical Use:** The findings strongly suggest that TRACE-Q is a more robust choice for applications requiring high query accuracy. The algorithm's ability to maintain a steady $F_1$-score at higher compression ratios could be particularly beneficial for scenarios where storage efficiency is critical without compromising query performance.

In conclusion, the experimental results presented in Figure 7 highlight the advantages of TRACE-Q in terms of query accuracy for both range and KNN queries. Future work should focus on extending the range of compression ratios tested to fully explore the capabilities and limitations of TRACE-Q in comparison to MRPA. This would help in delineating the specific conditions under which each algorithm performs optimally, thereby guiding users in selecting the most appropriate simplification method for their needs.

## VI. Conclusion and Future Work

In this paper, we have introduced a new query-driven approach to trajectory simplification, that aims to minimize query accuracy loss when querying the simplified trajectories compared

to querying the original trajectories. Our approach, TRACE-Q, utilizes the EDTS algorithm MRPA alongside a query accuracy evaluation process to produce simplified trajectories that fit within a satisfactory $F_1$-score. Our experiments show TRACE-Q outperforming MRPA in both range query accuracy and KNN query accuracy.

For future studies, it may be insightful to consider more types of spatial queries when determining the query accuracy of TRACE-Q such as Window Trajectory Distance Join and Window Trajectory KNN Join [14].

Our experiments only check the accuracies of range- and KNN queries. This is in part due to TRACE-Q using the same queries in the simplification process. However, it might be insightful to check whether TRACE-Q works for other types of spatial queries.

## VII. Acknowledgements

This work is carried out as a semester project on the second semester of the Software MSc at Aalborg University. The project is supervised by Tianyi Li, as an invaluable source of knowledge and recommendations throughout the project.

## References

[1] Yu Zheng. Trajectory Data Mining: An Overview. *ACM Transactions on Intelligent Systems and Technology*, 6 (3):1–41, May 2015. ISSN 2157-6904, 2157-6912. doi: 10.1145/2743025. URL https://dl.acm.org/doi/10.1145/2743025.

[2] Yanwei Yu, Lei Cao, Elke A. Rundensteiner, and Qin Wang. Detecting moving object outliers in massive-scale trajectory streams. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 422–431, New York New York USA, August 2014. ACM. ISBN 978-1-4503-2956-9. doi: 10.1145/2623330.2623735. URL https://dl.acm.org/doi/10.1145/2623330.2623735.

[3] Zhenhui Li, Jiawei Han, Ming Ji, Lu-An Tang, Yintao Yu, Bolin Ding, Jae-Gil Lee, and Roland Kays. MoveMine: Mining moving object data for discovery of animal movement patterns. *ACM Transactions on Intelligent Systems and Technology*, 2(4):1–32, July 2011. ISSN 2157-6904, 2157-6912. doi: 10.1145/1989734.1989741. URL https://dl.acm.org/doi/10.1145/1989734.1989741.

[4] Zheng Wang, Cheng Long, Gao Cong, and Christian S. Jensen. Collectively Simplifying Trajectories in a Database: A Query Accuracy Driven Approach, December 2023. URL http://arxiv.org/abs/2311.11204. arXiv:2311.11204 [cs].

[5] Dongxiang Zhang, Mengting Ding, Dingyu Yang, Yi Liu, Yu Fan, and Heng Tao Shen. Trajectory Simplification: An Experimental Study and Quality Analysis. *Proceedings of the VLDB Endowment*, 11(9):934–946, May 2018. doi: 10.14778/3213880.3213885. URL https://dl.acm.org/doi/epdf/10.14778/3213880.3213885.

[6] Minjie Chen, Mantao Xu, and Pasi Franti. A Fast O(N) Multiresolution Polygonal Approximation Algorithm for GPS Trajectory Simplification. *IEEE Transactions on Image Processing*, 21(5):2770–2785, May 2012. ISSN 1941-0042. doi: 10.1109/TIP.2012.2186146. URL https://ieeexplore.ieee.org/document/6142066. Conference Name: IEEE Transactions on Image Processing.

[7] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 99–108, San Jose California, November 2010. ACM. ISBN 978-1-4503-0428-3. doi: 10.1145/1869790.1869807. URL https://dl.acm.org/doi/10.1145/1869790.1869807.

[8] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–324, San Diego California USA, August 2011. ACM. ISBN 978-1-4503-0813-7. doi: 10.1145/2020408.2020462. URL https://dl.acm.org/doi/10.1145/2020408.2020462.

[9] Cheng Long, Raymond Chi-Wing Wong, and H. V. Jagadish. Direction-preserving trajectory simplification. *Proceedings of the VLDB Endowment*, 6(10):949–960, August 2013. ISSN 2150-8097. doi: 10.14778/2536206.2536221. URL https://dl.acm.org/doi/10.14778/2536206.2536221.

[10] Richard Bellman. On the approximation of curves by line segments using dynamic programming. *Communications of the ACM*, 4(6):284, June 1961. ISSN 0001-0782, 1557-7317. doi: 10.1145/366573.366611. URL https://dl.acm.org/doi/10.1145/366573.366611.

[11] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, December 1973. ISSN 0317-7173. doi: 10.3138/FM57-6770-U75U-7727. URL https://utpjournals.press/doi/abs/10.3138/FM57-6770-U75U-7727. Publisher: University of Toronto Press.

[12] John Hershberger and Jack Snoeyink. Speeding Up the Douglas-Peucker Line-Simplification Algorithm. Technical Report, University of British Columbia, CAN, March 1992.

[13] Nirvana Meratnia and Rolf A. de By. Spatiotemporal Compression Techniques for Moving Point Objects. In Elisa Bertino, Stavros Christodoulakis, Dimitris Plexousakis, Vassilis Christophides, Manolis Koubarakis, Klemens Böhm, and Elena Ferrari, editors, *Advances in Database Technology - EDBT 2004*, pages 765–782, Berlin, Heidelberg, 2004. Springer. ISBN 978-3-540-24741-8. doi: 10.1007/978-3-540-24741-8_44.

[14] Yun Chen and Jignesh M. Patel. Design and evaluation of trajectory join algorithms. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 266–

275, Seattle Washington, November 2009. ACM. ISBN 978-1-60558-649-6. doi: 10.1145/1653771.1653809. URL https://dl.acm.org/doi/10.1145/1653771.1653809.