Trajectory Simplification: An Experimental Study and Quality Analysis

Dongxiang Zhang[‡] Mengting Ding[‡] Dingyu Yang[§] Yi Liu[‡] Ju Fan[†] Heng Tao Shen[‡*]
[‡] Center for Future Media and School of Computer Science &Engineering, UESTC, China
[§]School of Electronics and Information, Shanghai Dian Ji University, China
[†] the Key Lab of Data Engineering and Knowledge Engineering, Renmin University of China
‡{zhangdo,mengting,liuyi}@uestc.edu.cn §yangdy@sdju.edu.cn †fanj@ruc.edu.cn ‡shenhengtao@hotmail.com

ABSTRACT

The ubiquitousness of GPS sensors in smart-phones, vehicles and wearable devices has enabled the collection of massive volumes of trajectory data from tracing moving objects. Consequently, an unprecedented scale of timestamped GPS data has been generated and posed an urgent demand for an effective storage mechanism for trajectory databases. The mainstream compression technique is called trajectory simplification, that finds a subsequence to approximate the original trajectory and attempts to minimize the information loss under a distance measure. Even though various simplification algorithms have been proposed in the past decades, there still lacks a thorough comparison to cover all the state-of-the-art algorithms and evaluate their quality using datasets in diversified motion patterns. Hence, it still remains a challenge for GPS data collectors to determine a proper algorithm in a concrete application. In addition, almost the entire line of previous methods uses error-based metrics to evaluate the compression quality, while ignoring their usability in supporting spatio-temporal queries on top of the reduced database. To bridge these gaps, we conduct so far the most comprehensive evaluation on trajectory simplification techniques. We compare the performance of 25 algorithms in total using five real datasets in different motion patterns. According to the experimental findings, we present useful guidance for the selection or development of effective trajectory simplification algorithms.

PVLDB Reference Format:
Dongxiang Zhang, Mengting Ding, Dingyu Yang, Yi Liu, Ju Fan, and Heng Tao Shen. Trajectory Simplification: An Experimental Study and Quality Analysis. *PVLDB*, 11 (9): 934-946, 2018.
DOI: https://doi.org/10.14778/3213880.3213885

1. INTRODUCTION

In recent years, the market has witnessed a sharp increase in the number of smart-phones, vehicles and wearable devices. With built-in GPS sensors, these devices have enabled many companies to collect an unprecedented scale of trajectory data by tracing the real-time positions of moving objects. Fitbit, one of the most popular wearable devices for fitness monitor and activity tracker, has collected the latest location data from its 23.2 million active users[1] at a high sampling rate. As another example, the ride-hailing giant Uber has hit 2 billion rides in June 2016[2], meaning that billion-scale vehicle trajectories have been acquired and call for an effective data management system for compressed storage, query processing and pattern discovery.

In the face of these massive volumes of trajectory data, an effective compression mechanism becomes a crucial component in the storage layer for a trajectory database. A straightforward idea is to exploit the redundancy in the spatial and temporal attributes to provide loseless compression. For instance, TrajStore [10] calculates the difference between two successive points $(x_{i-1}, y_{i-1}, t_{i-1})$ and (x_i, y_i, t_i) in each dimension and encodes the deltas using fixed point arithmetic. Consequently, the majority of points require a single byte for each delta. Trajic [36] calculates the delta values in a different way from TrajStore. It assumes the objects are moving with constant velocity and linear direction, and encodes the difference between p_i and p'_i, where p'_i is the predicted location from the previous point p_{i-1}. However, these algorithms have limited effect on storage reduction. As shown in Table 1, the compression ratios of TrajStore and Trajic in the five GPS datasets used in this paper are not encouraging.

Table 1: Compression ratios of lossless algorithms.

	Taxi	GeoLife	Truck	Illinois	Indoor
TrajStore	1.63	1.81	1.71	1.55	1.52
Trajic	1.74	2.69	2.05	2.80	1.77

Trajectory simplification has been by now the mainstream compression technique that provides much higher compression ratio with tolerable data loss. The idea is to discard redundant or less important points such that the original trajectory can be approximated by a series of successive line segments constructed from the remaining points. Existing algorithms fall into two categories based on the application modes. In the batch mode [2, 11, 16, 30, 31, 32], we are

*Corresponding Author: Heng Tao Shen

Proceedings of the VLDB Endowment, Vol. 11, No. 9

DOI: https://doi.org/10.14778/3213880.3213885

[1] http://expandedramblings.com/index.php/fitbit-statistics/
[2] http://www.webpronews.com/uber-hits-2-billion-rides-milestone-2016-07/

aware of the full historical data and aim at achieving a good balance between compression ratio and data loss at the server side. In the online mode [20, 26, 27, 32, 34, 35, 37, 39], only a local buffer is available at the sensor side and the goal is to maintain the most important points for the streaming GPS data. It is noticeable that an error bound ϵ can be enforced in both modes to ensure that the error of any discarded point does not exceed ϵ.

Semantic Compression [3, 21, 29, 38] is an alternative solution that leverages the knowledge of road networks to facilitate trajectory compression. The GPS points are first mapped to a road segment by any existing map-matching methods [17, 45]. The two dimensional locations (x_i, y_i) are then transformed into road segment ids and offsets. Since a local sequence of successive points can often be mapped to the same segment, the spatial redundancy is improved, which often leads to higher compression ratio. In addition, frequent travel paths in the road network can also be utilized to further reduce the storage cost. In this paper, the experimental evaluations of the semantic compression algorithms are beyond the scope for two main reasons. First, these methods require additional road network information which is not always available (e.g. in the indoor tracking applications [1]). Second, these compression methods are applied on top of the map-mapped points in the road segments, rather than the points in the original trajectory. It is possible that the moving objects are not confined to the well-defined tracks, resulting in significant deviation between the original data and the mapped data. Hence, it is difficult to provide a fair comparison with other compression methods functioned on the original trajectories.

In this paper, we aim at conducting a thorough evaluation for trajectory simplification which has been recognized as the mainstream solution to large-scale trajectory storage. Even though empirical studies on trajectory compression have been conducted before by Muckell et al. [33, 35] and Hunnik's work [41], they only cover a small number of algorithms. Since the compression technology has witnessed significant advancement in recent years, we believe there is a demand to conduct a new round of more comprehensive experimental analysis to show the recent progress.

Firstly, most of the algorithms evaluated in [33, 35] are heuristic and no longer state-of-the-art. There have been a considerable number of more advanced algorithms proposed afterwards, with more complex distance metric [8, 5] and processing logic [27, 28, 26]. There also emerges a new brach of direction-preserving simplification algorithms [19, 20, 30, 31], which claimed to guarantee both direction and position errors. In this paper, we will examine 25 trajectory simplification algorithms in total, covering both batch and online modes.

Secondly, even though various error-based metrics have been proposed to measure the compression quality, the usability of the reduced trajectory data still remains an unresolved question. In this paper, we make a reasonable assumption that in a trajectory database management system, the compression algorithms are located in the storage layer to support high-level query processing [13, 46] or pattern mining tasks [14, 44, 12]. Hence, data usability is also a key factor for compression quality measurement. In this paper, we are the first to empirically study the accuracies of supporting popular spatio-temporal queries on top of a compressed trajectory database.

To sum up, we make the following primary contributions in this paper:

1. We conduct so far the most comprehensive evaluation on trajectory simplification, covering 25 algorithms in total and 5 real datasets in different motion patterns. The evaluation covers four types of error metrics, including perpendicular Euclidean distance (PED), synchronized Euclidean distance (SED), direction error and speed error.
2. We propose to use the data usability of reduced trajectory database as an alternative performance indicator for compression quality and conduct the first experimental study on the accuracies of supporting range queries, kNN queries, spatial joins and trajectory clustering on top of simplified trajectories.
3. The codes are publicly accessible at Github[3].

The remaining of the paper is organized as follows. We present basic concepts and definitions about trajectory simplification in Section 2. The simplification algorithms in batch and online modes are presented and summarized in Section 3. Comprehensive experimental evaluation and analysis are conducted in Section 4. We conclude the paper, summarize the key observations, present future directions in Section 5.

2. PROBLEM DEFINITION

2.1 Basic Concepts of Trajectories

A trajectory $\mathcal{T}$ is a sequence of timestamped GPS data $p_1, p_2, \ldots, p_N$. The i-th point in $\mathcal{T}$, denoted by $\mathcal{T}[i]$, is represented as a triple (x_i, y_i, t_i), where (x_i, y_i) is a two-dimensional location and t_i records the sampling timestamp of p_i. We use $\mathcal{T}[t_s : t_e]$ to denote the **sub-trajectory** of $\mathcal{T}$ within time window $[t_s, t_e]$, which consists of points p_i with $s \leq i \leq e$. In contrast, a **subsequence** of $\mathcal{T}$ is represented as $p_{u_1}, p_{u_2}, \ldots, p_{u_M}$, where $p_{u_i} \in \mathcal{T}$ and $1 < u_1 < \ldots < u_M < N$. We use $\overrightarrow{p_s p_e}$ with $1 \leq s < e \leq N$ to denote a directed **segment** staring from point p_s and ending at point p_e. Without ambiguity, its direction is formally defined as:

Definition 1. *Direction of Segment $\overrightarrow{p_s p_e}$*
The direction of a segment $\overrightarrow{p_s p_e}$, denoted by $\theta(\overrightarrow{p_s p_e})$, is defined as the angle of an anticlockwise rotation from the positive x-axis to a vector from p_s to p_e.

The problem of trajectory simplification essentially finds a subsequence with M points ($M << N$) as the best approximation of the original trajectory $\mathcal{T}$. These M points split the id space $[1, N]$ into $M+1$ intervals. We call $\overrightarrow{p_{u_i} p_{u_{i+1}}}$ the **anchor segment** for the points located within the id interval $(p_{u_i}, p_{u_{i+1}})$. These points will be discarded, causing data loss; and the distance to the associated anchor segment is often used to measure compression error. In the following, we summarize the error metrics that have been proposed.

2.2 Error-Based Quality Metrics

We first present the definitions of three types of the most popular error metrics adopted by previous methods. For ease of presentation, we assume that p_m is a point to be discarded and $\overrightarrow{p_s p_e}$ is its anchor segment.

[3] https://github.com/uestc-db/traj-compression

DEFINITION 2. *Perpendicular Euclidean Distance (PED) The perpendicular Euclidean distance between p_m and its anchor segment $\overrightarrow{p_s p_e}$ is the shortest distance from p_m to line $\overrightarrow{p_s p_e}$ formally defined as*

$$PED(p_m) = \frac{|(y_e - y_s)x_m - (x_e - x_s)y_m + x_e y_s - y_e x_s|}{\sqrt{(y_e - y_s)^2 + (x_e - x_s)^2}}$$

DEFINITION 3. *Synchronized Euclidean Distance (SED) The synchronized Euclidean distance between the actual location p_m and its synchronized point $p'_m(x'_m, y'_m, t_m)$ on the anchor segment $\overrightarrow{p_s p_e}$ is defined as:*

$$SED(p_m) = \sqrt{(x_m - x'_m)^2 + (y_m - y'_m)^2}$$

where

$$\begin{aligned} x'_m &= x_s + \frac{x_e - x_s}{t_e - t_s}(t_m - t_s) \\ y'_m &= y_s + \frac{y_e - y_s}{t_e - t_s}(t_m - t_s) \end{aligned}$$

DEFINITION 4. *Direction-Aware Distance (DAD) We denote the directions of $\overrightarrow{p_m p_{m+1}}$ and $\overrightarrow{p_s p_e}$ as $\theta(\overrightarrow{p_s p_e})$ and $\theta(\overrightarrow{p_m p_{m+1}})$ respectively. The angular distance of segement $\overrightarrow{p_s p_e}$, denoted by $DAD(\overrightarrow{p_s p_e})$, is defined to be the greatest angular difference between the two directions. That is,*

$$DAD(\overrightarrow{p_s p_e}) = max_{s \le m < e} \triangle(\theta(\overrightarrow{p_s p_e}), \theta(\overrightarrow{p_m p_{m+1}}))$$

where

$$\begin{aligned} \triangle(\theta(\overrightarrow{p_s p_e}), \theta(\overrightarrow{p_m p_{m+1}})) = \min\{&|\theta(\overrightarrow{p_s p_e}) - \theta(\overrightarrow{p_m p_{m+1}})|, \\ &2\pi - |\theta(\overrightarrow{p_s p_e}) - \theta(\overrightarrow{p_m p_{m+1}})|\} \end{aligned}$$

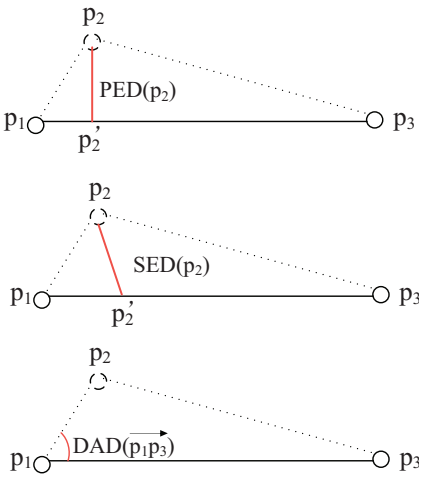


Figure 1: Examples of error-based metrics.

EXAMPLE 1. *Figure 1 illustrates examples of PED, SED and DAD. In these examples, we assume that points p_1 and p_3 are retained and p_2 is discarded, i.e., $\overrightarrow{p_1 p_3}$ is the anchor segment for p_2. Based on the definitions, $PED(p_2)$ is the shortest distance from p_2 to the segment whereas $SED(p_2)$ assumes that the object is moving at constant speed from p_1 to p_3 and p'_2 is the synchronized point of p_2 in segment $\overrightarrow{p_1 p_3}$. The angular distance $DAD(\overrightarrow{p_1 p_3})$ is defined as the angle between $\overrightarrow{p_1 p_2}$ and $\overrightarrow{p_1 p_3}$.*

Besides the three popular metrics, there exist other types of error metrics hand-crafted by previous methods. For instance, Chen et al. proposed integral square synchronous Euclidean distance (ISSD) [8] whose computation takes $O(1)$ time. A direction-based persistence (DBP) measure was proposed in [18] to capture the tolerable variation between local minima and maxima. In [25], position and velocity are considered as trajectory features worth preserving and velocity error is used to measure compression quality.

Given a selected error metric, an **error-bounded** simplification algorithm guarantees that the maximum error for the discarded points does not exceed the predefined threshold ϵ.

2.3 Query-Based Quality Measures

Previous simplification methods treat trajectory compression and storage as an independent component. However, in the bigger picture of a trajectory management system, trajectories are normally stored and indexed to support query processing or pattern mining. We observed that almost the entire line of previous simplification algorithms ignore the data usability of reduced trajectories when supporting querying and mining tasks in the higher layers. Thus, we propose to treat data usability as a key factor for quality measurement. In this following, we define three types of popular and representative spatio-temporal queries in a trajectory database, including range queries, kNN queries, spatial join and trajectory clustering, and use their accuracies in the reduced database as the **query-based quality measures** to be used in the experimental study.

DEFINITION 5. *Window Range Query (W-RQ) [4] Given a query cube $\langle q_{x_1}, q_{x_2}, q_{y_1}, q_{y_2}, q_{t_1}, q_{t_2} \rangle$, a W-RQ query finds all the trajectories with at least one point $p_i = (x_i, y_i, t_i)$ such that $q_{x_1} \le x_i \le q_{x_2}$, $q_{y_1} \le y_i \le q_{y_2}$ and $q_{t_1} \le t_i \le q_{t_2}$.*

To define kNN query in a trajectory database, we need to determine a proper distance measure between two trajectories. In the related literature, there have been several distance measures proposed, such as Dynamic Time Warping (DTW) [22], Edit distance with Real Penalty (ERP) [6], Longest Common Subsequences (LCSS) [42] and Edit Distance on Real Sequence (EDR) [7]. Among them, EDR has been well cited and widely adopted because it can reduce the effect of outliers and has the ability to handle local time shifting. In addition, it can assign penalties to gaps between two matched sub-trajectories and provide more accurate results. The formal definition of EDR is provided in the following:

$$EDR_{R,S}(i,j) = \begin{cases} 0, & i = j = 0 \\ i, & j = 0 \text{ and } i > 0 \\ j, & i = 0 \text{ and } j > 0 \\ \min \begin{cases} EDR_{R,S}(i-1, j) + 1 \\ EDR_{R,S}(i, j-1) + 1 \\ EDR_{R,S}(i-1, j-1) \\ +[!\text{match}(R_i, S_j)] \end{cases} & \text{else} \end{cases}$$

where R and S are two trajectories and the function `match` determines whether the distance of R_i and S_j is within the threshold ϵ. In this paper, we use EDR as the distance measure to define window-based kNN queries in a trajectory

database. We assume that the trajectories have been synchronized and the missing data have been interpolated.

DEFINITION 6. *Window kNN Query (W-kNN) [15]*
Given a query trajectory $\mathcal{T}_q$ within time window $[t_s, t_e]$ and a trajectory database $\mathcal{D}$, a W-kNN query returns a result set R with k trajectories such that for any $\mathcal{T}' \in R$ and $\mathcal{T}'' \in \mathcal{D} - R$, we have $EDR(\mathcal{T}[t_s : t_e], \mathcal{T}'[t_s : t_e]) \leq EDR(\mathcal{T}[t_s : t_e], \mathcal{T}''[t_s : t_e])$.

Finally, we define the spatial-join queries in a trajectory database.

DEFINITION 7. *Window Trajectory Distance Join (W-TDJ) [9]*
Given a query trajectory $\mathcal{T}_q$ within time window $[t_s, t_e]$ and a distance threshold δ, $\mathcal{T}'$ is a result for W-TDJ if for $t_s \leq i \leq t_e$, we have $dist(\mathcal{T}_q[i], \mathcal{T}'[i]) \leq \delta$, where $dist(\cdot, \cdot)$ represents Euclidean distance in this paper.

For the operator of trajectory clustering, we follow the definition proposed in [24]. It uses a tailored distance metric that takes into account perpendicular distance, parallel distance and angle distance. Interested readers can refer to [24] for more details.

3. SIMPLIFICATION ALGORITHMS

Trajectory simplification, in both online and batch mode, has been intensively studied in the past years. The objective is to reduce the original trajectory from N points to M points with $M << N$, while still preserving the significant positional or topological features. In the batch mode, the complete history of collected trajectories are available at the server side and a better trade-off between compression ratio and quality can be achieved with higher computation cost. In the online mode, we consider streaming applications and only a local buffer is available at the sensor side. Its goal is to maintain the most important points when buffer size is limited, so that communication overhead can also be reduced.

In the following, we categorize the algorithms into batch and online modes and summarize their features in Tables 2 and 3, respectively. Some of these algorithms are ad-hoc and some of them can guarantee that the compression ratio or distance-based error is bounded. In particular, there are several algorithms designed with optimality. They either find a sequence with exactly M points that minimize the compression error (called min-ϵ problem) or they guarantee the error is at most ϵ with the objective of minimizing the number of points retained (called min-# problem).

3.1 Trajectory Simplification in Batch Mode

Bellman [2] is considered as the first algorithm for trajectory simplification. It uses dynamic programming to find a subsequence with M points that generates the minimum spatial distance error. The time complexity of the exact algorithm is $O(N^3)$. After that, **DP** [11] was proposed as an approximate simplification method with error bound guarantee. Given a trajectory segment and an error threshold ϵ, it finds the point p' causing the greatest perpendicular distance d_s. If $d_s < \epsilon$, the approximation is accepted and we only need to keep the two end points and discard the remaining points within the segment. Otherwise, we split T into two sub-trajectories based on the selected point and recursively apply the process. The worst time complexity of **DP** is $O(N^2)$. To improve the efficiency of **DP** algorithm, **DPhull** [16] was proposed to take advantage of the properties of convex hull and save the cost of finding the point with the greatest perpendicular distance. The algorithm returns the same set of points as **DP** but reduces the complexity to $O(N \log N)$. **TD-TR** [32] is also an extension of **DP** by exploiting the temporal dimension. It uses the same algorithm framework as **DP**. The only difference is that a new distance measure named Synchronous Euclidean Distance (SED) was proposed to replace the perpendicular distance used in **DP** when finding the split point with the maximum distance.

The above methods attempt to preserve the positional information of original trajectories. In [8], the proposed **MRPA** algorithm uses a distance measure named integral square synchronous Euclidean distance (ISSD) with $O(1)$ computation. In [30], Long et al. observed that the simplified trajectories by position-preserving algorithms may lose critical information for trajectory clustering or online query processing. They proposed direction-preserving trajectory simplification (DPTS) strategies that considered angular distance measure and showed that DPTS not only preserves direction information, but also preserves position information. Based on the new distance measure, an algorithm named **SP** was proposed to solve the min-# problem with complexity $O(C \cdot N^2)$, where C is a small constant in many cases, and an approximate algorithm **Intersect** with $O(N)$ complexity and degraded quality. In their subsequent work [31], they attempted to solve the min-ϵ problem in the DPTS framework. Similarly, they proposed an exact algorithm named **Error-Search** with complexity $O(N^2 \log N)$ and an approximate algorithm named **Span-Search** with complexity $O(N \log^2 N)$.

3.2 Trajectory Simplification in Online Mode

In the online mode, GPS data are continuously sampled at the sensor side. These data are stored in a local buffer and an online simplification algorithm decides which points to drop. The remaining points will be sent to the remote server for further processing and the communication overhead can be reduced due to trajectory simplification. Since the online algorithms do not have the knowledge of the entire trajectory, they cannot achieve optimal results for the min-# and min-ϵ problems. The most straightforward solution is **Uniform** which samples at a fixed rate, i.e. points $p_1, p_{\lambda+1}, p_{2\lambda+1}, \ldots$ will be preserved. Despite the simplicity, it can even achieve promising performance in certain scenarios, as will be illustrated in the following experimental study.

OPW [32] is a very early algorithm designed for online simplification. As a new point arrives in the buffer, it uses the new point and the first point to build an anchor segment and calculates the PED distance for all the points in the buffer. If the maximum distance is larger ϵ, the new point is sampled; otherwise, the algorithm proceeds to the next incoming point. Its variant based on SED distance measure is named **OPW-TR**. The complexity of these two algorithms in the worst case is $O(N^2)$.

Dead Reckoning [39] reduces the time complexity to $O(N)$ as it incurs $O(1)$ cost in distance calculation for each incoming point. The algorithm assumes that each object is moving in a constant velocity and direction which can

Table 2: Summary of Trajectory Simplification Algorithms in Batch-Mode (sorted by publication year).

Algorithms	Time Cost	Space Cost	Size-Bounded	Error-Bounded	Error Criterion	Optimality
Bellman [2]	$O(N^3)$	$O(N^2)$	yes	no	PED	min-ϵ
DP [11]	$O(N^2)$	$O(N)$	no	yes	PED	no
DPhull [16]	$O(N \log N)$	$O(N)$	no	yes	PED	no
TD-TR [32]	$O(N^2)$	$O(N)$	no	yes	SED	no
MRPA [8]	$O(\frac{N^2}{M})$	$O(N)$	yes	no	ISSD	min-ϵ
SP [30]	$O(C \cdot N^2)$	$O(N)$	no	yes	DAD	min-#
Intersect [30]	$O(N)$	$O(N)$	no	no	DAD	no
Error-Search [31]	$O(N^2 \log N)$	$O(N^2)$	yes	no	DAD	min-ϵ
Span-Search [31]	$O(N \log^2 N)$	$O(N)$	yes	no	DAD	no

Table 3: Summary of Trajectory Simplification Algorithms in Online-Mode (sorted by publication year).

Algorithms	Time Cost	Space Cost	Size-Bounded	Error-Bounded	Error Criterion
Uniform	$O(N)$	$O(1)$	no	no	-
OPW [32]	$O(N^2)$	$O(N)$	no	yes	PED
OPW-TR [32]	$O(N^2)$	$O(N)$	no	yes	SED
Dead Reckoning [39]	$O(N)$	$O(1)$	no	yes	prediction error
Threshold [37]	$O(N)$	$O(1)$	no	yes	speed and direction
STTrace [37]	$O(N \cdot \log \beta)$	$O(\beta)$	yes	no	SED
SQUISH [34]	$O(N \cdot \log \beta)$	$O(\beta)$	yes	no	SED
CDR [23]	$O(N^2)$	$O(N)$	no	yes	PED
SQUISH-E(λ) [35]	$O(N \log \frac{N}{\lambda})$	$O(\beta)$	yes	no	SED
SQUISH-E(μ) [35]	$O(N \log N)$	$O(N)$	no	yes	SED
Persistence [18]	$O(N)$	$O(1)$	no	yes	DBP
BQS [27, 28]	$O(N^2)$	$O(N)$	no	yes	PED
FBQS [27, 28]	(N)	$O(1)$	no	yes	PED
Angular [20]	$O(N)$	$O(1)$	no	yes	DAD
Interval [19]	$O(N)$	$O(1)$	no	yes	DAD
DOTS [5]	$O(\frac{N}{\lambda})$	$O(\beta^2)$	yes	no	ISSD
OPERB [26]	$O(N)$	$O(1)$	no	yes	PED

be derived from the recent historical data. This study calculates the Euclidean distance between each incoming point and its predicted counterpart and uses the distance to determine whether the new point should be preserved or dropped. It was proved that the method has an error bound when applied in the batch mode to simplify the entire trajectory. **CDR** [23] is similar to **Dead Reckoning** when applied in the moving object tracing applications. It refines the preservation condition of a sampled point in order to achieve a higher compression ratio and guarantee the same error bound as **Dead Reckoning**. However, its computation cost is $O(N^2)$ in the worst case.

Threshold [37] and **Persistence** [18] do not apply Euclidean distance measure when deciding whether a point should be preserved. Instead, **Threshold** [37] poses two threshold constraints on both speed and direction variation. Its compression ratio is not high because a point is discarded only when both constraints are satisfied. **Persistence** [18] introduces a new heuristic approach based on topological persistence to maintain the sharp directional features of trajectories. It produces compact high quality approximations and runs in linear complexity.

When the local buffer is of fixed size (say β), a variant named **STTrace** was proposed in [37] to guarantee that at most β points will be preserved. If the buffer is full when a new point arrives, the algorithm will pick the point in the buffer with the minimum SED distance and evict it. **SQUISH** [34] also works in the streaming environment with a fixed-size buffer. For each point p_m, it maintains a priority $\pi(p_m)$ which serves as the upper bound of SED distance for the neighboring points. This is because when a point is deleted, its priority score will be accumulated to its two neighboring point. Since **SQUISH** is not error-bounded, its subsequent work **SQUISH-E** [35] was designed to be adaptive to different objectives by introducing two parameters λ (used as compression ratio bound) and μ (used as compression error bound). When setting $\mu = 0$, the algorithm minimizes SED error ensuring the compression ratio of λ. We denote the algorithm **SQUISH-E(λ)**. When $\lambda = 1$, the algorithm maximizes compression ratio while keeping errors under μ, denoted by **SQUISH-E(μ)**.

BQS [27, 28] picks at most eight significant points, forming a convex hull to enclose all the points in the buffer. Then, an upper bound and a lower bound are derived such that in most cases, a point can be quickly decided for removal or preservation with cost $O(1)$. However, the running time of **BQS** still remains $O(N^2)$ in the worst case. **FBQS** [27, 28] is a fast version of **BQS** [27] that avoids deviation calculation and eliminates the necessity of maintaining a buffer. Consequently, the time complexity

is reduced to $O(N)$ but more points will be preserved in **FBQS** than in **BQS**. In [28], the amnesic framework of BQS, named ABQS, is proposed to specifically handle aging trajectories and its evaluation is beyond the scope of this paper. In [26], Liu et al. proposed a one-pass error bounded trajectory simplification algorithm named **OPERB**. It is based on a local distance checking method and maintains a directed line segment to approximate the buffered points and the distance from the current point to the line segment must be bounded. Five optimization techniques were proposed to further improve the compression ratio. The time complexity is $O(N)$ and space cost is $O(1)$.

The works of **Angular** [19] and **Interval** [20] belong to another branch of online simplification that works in the direction-preserving scenario. **Angular** maintains a variable ϵ_a to accumulate the angular difference of the points recently arrived. If ϵ_a is smaller than the threshold, it preserves the latest point and resets ϵ_a. However, such an upper bound estimation is too loose and an improved version **Interval** was proposed to discard more points and still guarantee the error bound. Both algorithms achieve $O(N)$ time complexity and $O(1)$ space overhead.

DOTS [5] can be viewed as an online version of **MRPA** [8], with the same type of distance measure. It constructs an incremental DAG (directed acyclic graph) with multiple layers for online points. The ISSED will be locally minimized by adjusting edges between consecutive layers and one of the shortest paths will be selected as the simplified trajectory.

4. EXPERIMENTAL ANALYSIS

In this section, we evaluate the performance of 25 trajectory simplification algorithms with 5 real datasets in different motion patterns. We implement in C/C++ the algorithms of **DP** and **TD-TR** in the batch mode and **Uniform**, **OPW**, **OPW-TR**, **Dead Reckoning**, **Threshold**, **Persistence**, **STTrace**, **SQUISH-E(λ)**, **SQUISH-E(μ)** and **Persistence** in the online mode. The source codes of remaining algorithms are generously provided by the authors. All the experiments are conducted on a server with 6TB disk space, 40 CPU cores (Intel Xeon CPU E5-2650 with 2.30GHz) and 256GB memory.

4.1 Datasets

Since the characteristics of a GPS traces could differ substantially based on the transportation mode of moving objects, we use 5 types of trajectory datasets to examine whether an algorithm is robust to different motion patterns. Statistics of these datasets are shown in Table 4.

- GeoLife[4]: The dataset essentially keeps all the travel records of 182 users for a period of over three years, including multiple kinds of transportation modes (walking, driving and taking public transportation). The trajectories are sampled every $1 \sim 5$ seconds, with an average speed of 5.73m/s between two neighbor points.
- Taxi [43]: The dataset tracks the trajectories of $15,054$ taxies in Singapore. For each taxi, the GPS information are continually collected for one entire month with the sampling rates from half a minute to three minutes. Its average distance between two neighbor points is much higher than that in GeoLife.
- Indoor[5]: The dataset contains trajectories of visitors in the ATC shopping center in Osaka. To better capture the indoor activities, its maximum filter update rate is fixed to 30Hz and the visitor locations are sampled every $0.03 \sim 0.06$ second.
- Truck[6] [26]: The dataset is the GPS trajectories collected by trucks equipped with GPS sensors in China during a period from Aug. 2015 to Oct. 2015. The sampling rate varied from 3s to 60s. It's average speed is comparable to that in Taxi, but with a smaller average distance. The reason is that the trucks have higher speed when moving on highways, but they may rest at the same location for a long time.
- Illinois[7]: The dataset is obtained from two members in Argonne National Laboratory of UIC during their daily commute for 6 months. Each trajectory represents a continuous trip of a member in the Cook county and/or the Dupage county of Illinois. The trajectories are sampled strictly every second. The dataset has the lowest average distance, i.e., the highest redundancy.

Table 4: Statistics of datasets.

	Number of trajectories	Number of points	Sampling rate	Average speed	Average distance
GeoLife	36,479	24,178,070	1s∼5s	5.73m/s	9.96m
Taxi	574,329	350,811,106	21s∼193s	4.86m/s	280.04m
Indoor	529,397	330,117,253	0.03s∼0.06s	0.79m/s	0.037m
Truck	10,110	10,059,685	3s∼60s	4.32m/s	82.74m
Illinois	459	355,968	1s	0.008m/s	0.008m

4.2 Compression Time Analysis

Table 5: Compression time per trajectory point (in μs) for all the algorithms.

Algorithms	Lang	Taxi	Truck	GeoLife	Indoor
Batch Methods					
DP [11]	C/C++	0.53	0.46	0.46	0.52
DPhull [16]	C/C++	0.15	0.15	0.16	0.19
TD-TR [32]	C/C++	0.50	0.45	0.50	0.65
MRPA [8]	Matlab	18.57	24.29	19.49	20.06
SP [30]	C/C++	5.69	15.67	15.00	19.23
Intersect [30]	C/C++	0.28	0.28	0.28	0.30
Error-Search [31]	C/C++	664.29	945.02	652.65	651.99
Span-Search [31]	C/C++	28.29	35.07	32.27	28.42
Online Methods					
Uniform	C/C++	0.012	0.009	0.012	0.011
OPW [32]	C/C++	1.46	5.02	1.66	0.81
OPW-TR [32]	C/C++	1.72	6.89	1.42	0.87
DR [39]	C/C++	0.32	0.25	0.29	0.32
Threshold [37]	C/C++	1.16	1.04	1.07	0.60
STTrace [37]	C/C++	4.63	6.41	5.47	5.37
SQUISH [34]	C/C++	4.44	5.95	5.08	5.13
CDR [23]	C/C++	1.75	4.54	1.84	4.05
SQUISH-E(λ) [35]	C/C++	2.55	3.09	3.03	2.91
SQUISH-E(μ) [35]	C/C++	22.35	29.59	26.78	26.79
Persistence [18]	C/C++	0.54	0.50	0.49	0.65
BQS [27, 28]	Python	486.93	500.91	739.49	596.73
FBQS [27, 28]	Python	357.90	405.38	451.08	411.60
Angular [20]	Java	0.21	0.20	0.22	0.23
Interval [19]	Java	0.31	0.28	0.29	0.31
DOTS [5]	C/C++	1.61	2.67	1.58	1.86
OPERB [26]	Java	0.84	0.97	0.90	0.94

[4] http://research.microsoft.com/en-us/projects/geolife/

[5] http://www.irc.atr.jp/crest2010_HRI/ATC_dataset/

[6] http://mashuai.buaa.edu.cn/traj.html

[7] https://www.cs.uic.edu/~boxu/mp2p/gps_data.html

Table 5 shows the average CPU time to process a trajectory point in the batch and online modes. **DPhull** is an improved version of **DP** and runs much faster than **DP**. The performance of TD-TR is close to DP because they share the same processing logic but with different distance metrics. **MRPA** is much slower than **TD-TR** as its accumulative distance measure requires higher computation cost. In the direction-aware batch approaches, **Error-Search** requires $O(N^2 \log N)$ complexity and incurs the highest running time. In the online methods, the results of **Uniform** can be seen the lower bound of processing time as it involves only the simplest processing logic. **SQUISH-E(λ)** requires much more processing time than the two other variants because after compression by **SQUISH**, it iteratively picks the point with the minimum priority, removes it if the compression ratio can still be guaranteed, and adjusts the items in the priority queue. The running time of **BQS** and **FBQS** are much higher than the other methods for two main reasons. First, they involve several complex operators such as determining the quadrant and estimating the lower and upper bounds. Second, these two algorithms provided by the authors were implemented in Python. **FBQS** is slightly faster than **BQS** as it avoids the deviation operator which is $O(N)$ in the worst case.

4.3 Compression Error Analysis

To evaluate the quality of batch algorithms, we fix their compression ratio as 10 and examine their compression errors under different types of distance measures. The results of PED, SED, DAD and Speed errors are reported in Table 6. Among these algorithms, the best four in each column are marked in bold for the visualization purpose so that readers can quickly identify the promising algorithms among various datasets and error metrics.

The most important finding is that each algorithm is designed for a particular distance measure and its objective function is to minimize the error on that metric. Therefore, these algorithms show biased performance on different error metrics. A typical example is the direction-aware compression family (including **SP**, **Intersect**, **Error-Search** and **Span-Search**). They perform remarkably well in the metric of DAD, but cannot well preserve the location and speed information. Another example is **TD-TR**, which is a simple extension of **DP** algorithm by applying SED metric when choosing the split point. We can see that it is significantly superior over **DP** in the SED columns but shows worse performance when the metric is PED. Other observations that are worth noting are summarized in the following. 1) The datasets of Indoor and Illinois are sampled at a high frequency and the compression errors are much smaller in these two datasets with high redundancy. 2) The simplified trajectories by **DP** and **DPhull** are identical and thus they obtain the same compression errors in the table. Moreover, these two algorithms yield the smallest PED errors in all the datasets since they are particularly designed for this metric. 3) **MRPA** and **TD-TR** are two algorithms designed for synchronized distance errors. The former algorithm uses bottom-up merge strategy while the latter one applies iterative top-down split. These two algorithms work particularly well in the SED metric.

For the online methods, we also fix the compression ratio as 10 and report the results of four types of errors in Table 7. For better presentation, we mark the best five results in each column in bold. We observe that 1) Again, most of the algorithms exhibit biased performance on different error metrics as they are optimized for a particular distance measure. **Angular** and **Interval** are designed for DAD and only work well in this distance error. **OPW-TR** is a variant of **OPW** by replacing PED metric with SED. Thus, it is superior in the SED metric but works worse than **OPW** in the PED metric. 2) **DOTS** outperforms the other methods in the metrics of PED, SED and Speed. It uses ISSD as the distance metric and can be seen as an online version of **MRPA**. The superiority of **DOTS** and **MRPA** implies that an appropriate and robust distance measure can play a key role for compression quality. The ISSD cumulates the SED errors of a local segment. Even though higher calculation cost is required, it helps reduce the errors under multiple metrics. 3) **Threshold** poses two constraints on both speed and direction. After adjusting the thresholds to obtain the required compression ratio, many key information has been lost. Thus, its performance is rather unsatisfactory. 4) **Persistence** is also a heuristic that focuses on preserving the sharp features in the shape. It does not work well for PED, SED and Speed. 5) **STTrace** and the **SQUISH** family share similar processing logic, by evoking the one with the minimum SED error or the smallest priority score when a new point arrives. These algorithms obtain comparable errors in the four metrics. Overall, **SQUISH-E(μ)** is the best among them because its objective is to minimize the number of simplified points in the meanwhile keeping the error below the threshold. With the same compression ratio, it allows a smaller error threshold. 6) When considering SED metric in the Truck dataset, the algorithms design for metric other than SED tend to have SED errors higher than 1000. The reason is that in the Truck dataset, the trucks could stay at the same location for a long period. The methods without using SED in the optimization function are likely to preserve the first point. The remaining redundant points at the same location do not incur PED error but generate a considerable amount of SED error.

4.4 Compression Errors w.r.t Varying Ratios

In this set of experiments, we evaluate the compression errors w.r.t. varying size of simplified trajectory database. We use Taxi and GeoLife as two representative datasets and report the results of batch algorithms in Figure 2 and online algorithms in Figure 3. For each simplification mode, we selectively plot the algorithms whose errors are marked bold in Tables 6 and 7. The algorithms demonstrate similar patterns in the Taxi and GeoLife datasets, i.e., an algorithm works best in Taxi is likely to outperform others in the GeoLife dataset.

In the batch mode, when PED is used, the compression error of **MPRA** gets close to that of **DP** as the compression ratio ascends. When the ratio reaches 27, its performance can be even slightly better than **DP**. This implies that **MPRA** is more suitable for a trajectory database to be fiercely compressed. In addition, **MPRA** also outperforms the other methods in the metrics of SED and Speed, inferring that it is a robust solution in multiple error metrics. **DP**, though working good in PED, scales poorly in SED. Its SED errors increase dramatically as the compression ratio grows. **Error-Search** achieves the best performance in the direction metric because it guarantees the minimum compression error under a fixed compression ratio. **MRPA**

Table 6: Error results for batch algorithms with the same compression ratio ($ratio = 10$).

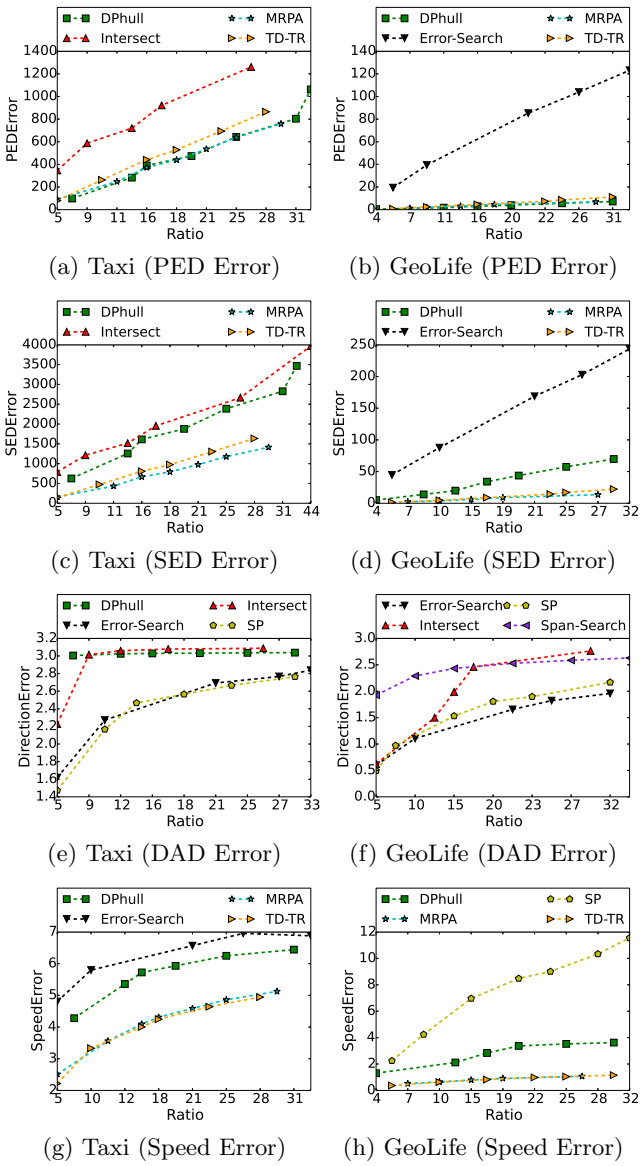
	PED Errors					SED Errors				
	Taxi	Truck	GeoLife	Indoor	Illinois	Taxi	Truck	GeoLife	Indoor	Illinois
DPhull [16]	**218.36**	**29.50**	**1.64**	**0.015**	**0.00050**	**1015.61**	**1162.12**	**20.26**	**0.041**	0.077
TD-TR [32]	**264.86**	**40.36**	**2.40**	**0.019**	**0.00071**	**478.12**	**79.16**	**4.53**	**0.031**	**0.0015**
MRPA [8]	**218.64**	**30.31**	**2.02**	**0.016**	**0.00059**	**379.97**	**56.20**	**3.79**	**0.027**	**0.0012**
SP [30]	794.30	1761.54	216.13	0.20	0.015	1420.57	3019.28	360.13	0.25	0.039
Intersect [30]	**619.77**	1531.02	186.17	0.16	0.0098	**1279.02**	2792.14	668.74	0.20	0.083
Error-Search [31]	803.06	1191.48	**33.73**	0.061	**0.0045**	1404.22	2221.03	**80.88**	0.091	**0.015**
Span-Search [31]	659.04	**940.32**	42.47	**0.049**	0.014	1362.84	**2173.16**	121.77	**0.079**	**0.048**
	Direction Errors					Moving Speed Errors				
	Taxi	Truck	GeoLife	Indoor	Illinois	Taxi	Truck	GeoLife	Indoor	Illinois
DPhull [16]	**3.04**	2.93	2.46	2.82	2.68	**5.09**	**74.27**	**1.67**	**0.33**	0.010
TD-TR [32]	3.05	**2.89**	2.49	2.78	2.60	**3.35**	**2.36**	**0.63**	**0.29**	**0.00043**
MRPA [8]	3.10	2.90	2.64	2.88	2.68	**3.41**	**3.82**	**0.65**	**0.30**	**0.00044**
SP [30]	**2.22**	**1.99**	**0.97**	**1.77**	**0.35**	5.86	86.60	**3.45**	0.36	**0.0013**
Intersect [30]	**3.03**	2.99	**1.26**	**2.46**	**0.38**	5.95	93.99	5.10	**0.35**	0.011
Error-Search [31]	**2.17**	**1.86**	**0.94**	**1.51**	**0.31**	**5.81**	**85.21**	10.65	0.37	**0.0010**
Span-Search [31]	3.05	**2.84**	**2.18**	**1.99**	**1.19**	5.87	94.88	12.03	0.36	**0.0052**

Table 7: Error results for online algorithms with the same compression ratio ($ratio = 10$).

	PED Errors					SED Errors				
	Taxi	Truck	GeoLife	Indoor	Illinois	Taxi	Truck	GeoLife	Indoor	Illinois
Uniform	345.97	159.81	8.45	0.023	0.0012	760.57	410.69	30.87	0.041	0.024
OPW [32]	**288.61**	**50.16**	**2.55**	**0.018**	0.00080	1134.05	1245.25	26.26	0.042	0.16
OPW-TR [32]	289.27	79.31	3.55	0.022	0.00095	**554.41**	**164.72**	**7.19**	**0.037**	**0.0022**
Dead Reckoning [39]	392.22	115.88	3.89	0.021	0.00079	1001.48	1040.67	19.82	0.042	0.0057
Threshold [37]	1040.73	680.79	286.08	0.36	0.057	2344.61	2033.78	652.98	0.45	0.15
STTrace [37]	402.15	124.91	8.83	0.038	0.0015	786.94	408.58	28.07	0.061	0.0035
SQUISH [34]	353.76	144.42	10.26	0.025	0.00098	**784.45**	426.29	28.64	0.044	**0.0025**
CDR [23]	387.22	64.74	5.06	0.11	0.0019	896.33	**166.19**	**11.07**	0.18	0.0055
SQUISH-E(λ) [35]	332.58	113.99	7.35	0.023	0.0011	**676.59**	**370.55**	22.83	**0.040**	**0.0023**
SQUISH-E(μ) [35]	**283.06**	**38.29**	**2.55**	**0.020**	**0.00077**	**530.11**	**81.40**	**5.01**	**0.033**	**0.0017**
Persistence [18]	326.53	69.43	103.71	0.029	0.022	1154.11	1334.62	164.06	0.060	0.093
BQS [27, 28]	**274.75**	**38.23**	**2.29**	**0.020**	**0.00068**	1015.60	1159.56	21.15	0.045	0.032
FBQS [27, 28]	**260.06**	**36.63**	**2.17**	**0.017**	**0.00057**	959.92	1147.70	**14.21**	**0.040**	0.013
Angular [20]	882.79	1532.65	284.47	0.19	0.016	1831.90	3219.05	804.26	0.25	0.042
Interval [19]	1240.72	1889.81	274.29	0.32	0.015	2489.10	5090.83	450.04	0.39	0.039
DOTS [5]	**228.91**	**32.95**	**2.18**	**0.016**	**0.00056**	**366.52**	**60.72**	**4.10**	**0.027**	**0.0011**
OPERB [26]	326.92	306.20	2.73	**0.019**	**0.00068**	1179.36	1577.67	21.01	0.042	0.085
	Direction Errors					Moving Speed Errors				
	Taxi	Truck	GeoLife	Indoor	Illinois	Taxi	Truck	GeoLife	Indoor	Illinois
Uniform	3.06	3.01	2.75	2.88	2.82	5.53	95.65	12.00	0.38	0.010
OPW [32]	3.06	2.93	2.58	2.83	2.78	5.02	60.61	2.08	**0.34**	0.012
OPW-TR [32]	**3.05**	**2.91**	2.59	2.80	2.75	**3.62**	**4.89**	**0.76**	**0.31**	**0.00055**
Dead Reckoning [39]	3.06	2.99	2.69	2.87	2.83	5.08	78.07	6.46	0.37	0.00091
Threshold [37]	3.06	2.92	2.80	2.86	2.74	5.77	44.68	3.85	0.39	0.010
STTrace [37]	3.06	2.92	2.58	**2.76**	**2.63**	**3.58**	**27.96**	9.61	**0.29**	**0.00051**
SQUISH [34]	3.06	2.92	2.66	2.86	2.72	4.32	39.76	11.26	0.36	**0.00062**
CDR [23]	3.06	**2.91**	2.66	2.88	2.81	**4.21**	**4.80**	**0.95**	0.41	0.00081
SQUISH-E(λ) [35]	3.06	2.94	2.65	2.82	2.72	4.29	51.40	11.02	0.35	0.00060
SQUISH-E(μ) [35]	**3.05**	**2.90**	**2.55**	**2.79**	**2.65**	**3.42**	**4.05**	**0.69**	**0.30**	**0.00046**
Persistence [18]	**3.03**	2.98	2.69	**2.78**	**2.69**	5.38	112.33	17.35	0.40	0.010
BQS [27, 28]	3.06	**2.91**	**2.47**	2.86	2.75	4.91	75.14	1.97	0.36	0.0044
FBQS [27, 28]	3.06	**2.91**	2.50	2.85	2.75	4.87	75.41	**1.90**	0.36	0.0026
Angular [20]	**3.05**	3.00	**2.30**	**2.72**	**0.58**	6.01	80.38	5.37	0.36	0.0013
Interval [19]	3.06	3.04	**1.32**	**2.31**	**0.39**	6.10	88.40	3.68	0.36	0.0013
DOTS [5]	**3.05**	**2.90**	2.60	2.83	2.70	**3.52**	**3.95**	**0.70**	**0.32**	**0.00046**
OPERB [26]	3.08	2.99	**2.47**	2.82	2.71	5.30	83.76	2.85	0.35	0.011

and **TD-TR** perform significantly better than **DP** in the Speed metric as they take into account the temporal attribute.

In the online mode, **DOTS** has a clear advantage in the metrics of PED and SED. It also demonstrates good scalability w.r.t. compression ratio. We can see that as ratio



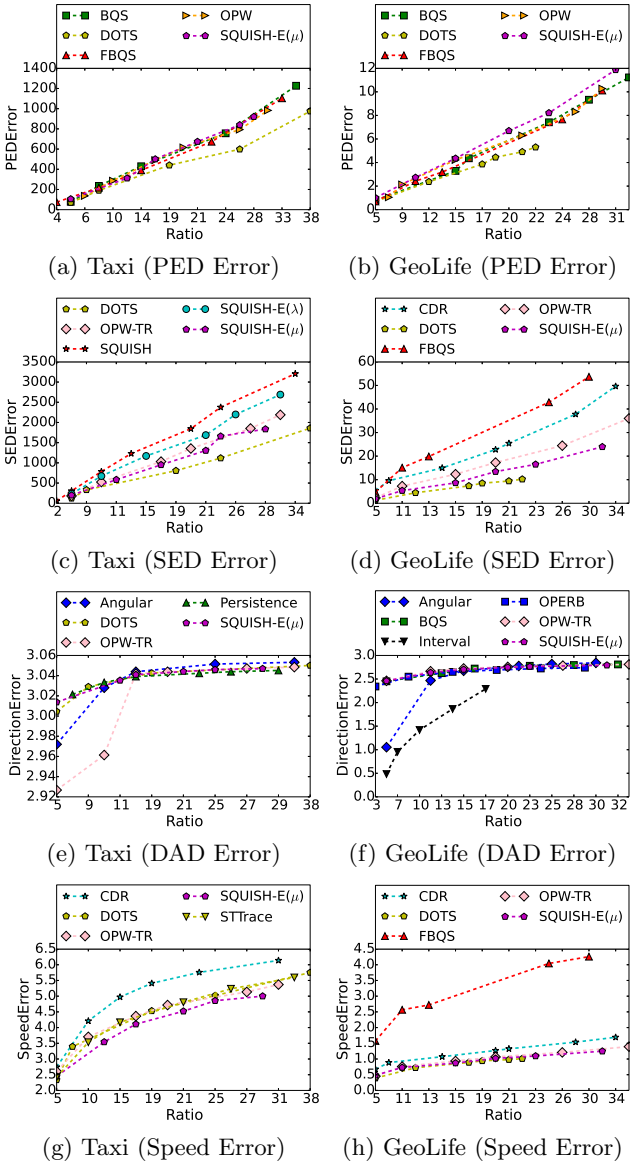
(a) Taxi (PED Error) (b) GeoLife (PED Error)

(c) Taxi (SED Error) (d) GeoLife (SED Error)

(e) Taxi (DAD Error) (f) GeoLife (DAD Error)

(g) Taxi (Speed Error) (h) GeoLife (Speed Error)

Figure 2: Error-ratio variation in the batch mode.



(a) Taxi (PED Error) (b) GeoLife (PED Error)

(c) Taxi (SED Error) (d) GeoLife (SED Error)

(e) Taxi (DAD Error) (f) GeoLife (DAD Error)

(g) Taxi (Speed Error) (h) GeoLife (Speed Error)

Figure 3: Error-ratio variation in the online mode.

increases, the performance gap is widened. In the direction-based error metric, **Interval** is dominating other methods with much smaller errors. However, it shows poor scalability w.r.t. compression ratio and its errors increase sharply as the ratio grows. As to the Speed metric, **SQUISH-E(μ)** and **OPW-TR** achieve comparable performance to **DOTS** and they also scale well with compression ratio.

4.5 Database Usability

4.5.1 *Evaluation Metric*

Given a W-RQ query, let R_g denote the result returned by the original trajectory database and R_c denote the trajectories returned by the compressed database. The precision of W-RQ is defined as

$$prec(\text{W-RQ}) = \frac{|R_c \cap R_g|}{|R_c|} \tag{1}$$

The recall of W-RQ is defined as

$$recall(\text{W-RQ}) = \frac{|R_c \cap R_g|}{|R_g|} \tag{2}$$

The accuracy of supporting W-RQ is defined as the F1-measure by incorporating $prec$(W-RQ) and $recall$(W-RQ). The accuracy of W-TDJ is defined the same way as W-RQ because $|R_c|$ is not necessarily equal to $|R_g|$. For W-kNNQ query, we define the accuracy as

$$accu(\text{W-kNNQ}) = \frac{|R_c \cap R_g|}{k} \tag{3}$$

To define the accuracy of trajectory clustering, we assume that the trajectories in the original and simplified datasets are both partitioned into m clusters via the clustering method proposed in [24]. Let C_O denote the clustering results derived from the original trajectory database. Given a pair of trajectories $\mathcal{T}_i$ and $\mathcal{T}_j$, if they appear in the same cluster, we consider $(\mathcal{T}_i, \mathcal{T}_j) \in C_O$. Similarly, we can define the clustering results of C_S for the simplified trajectories. Then, we can formally determine the precision and recall of trajectory clustering and use them to define the F1-measure:

$$prec(\text{clustering}) = \frac{|C_O \cap C_S|}{|C_S|} \quad (4)$$

$$recall(\text{clustering}) = \frac{|C_O \cap C_S|}{|C_O|} \quad (5)$$

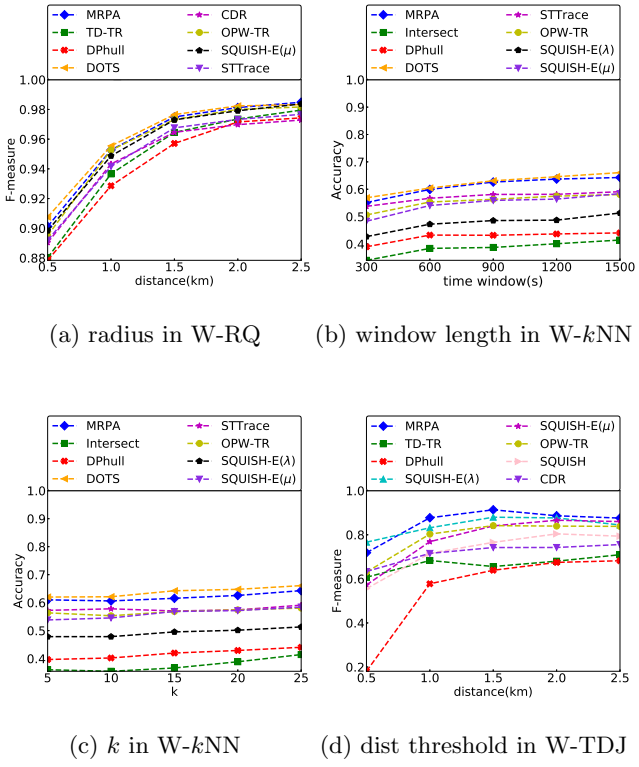
4.5.2 Experimental Setting

In the default settings, we use a square with radius set to 2km to specify the spatial constraint. The number of k in kNN query is set to 25. In the join query, the window length is 25 minutes and the distance threshold is 2km. To measure the accuracy of trajectory clustering, we run the TRACLUS algorithm [24] with the code provided by the authors. For fair comparison, we tune the parameters to uniformly generate 80 clusters for every dataset. We do not use Indoor dataset in the following experiments because its average time span of trajectory is a few seconds and not adequate to test the W-TDJ query. We also neglect the Illinois dataset because it is highly redundant. There is little differentiation for the query metrics among the simplification methods as their accuracies are always close to 1. With the simplified datasets by the compression methods, we conduct a linear interpolation to align the GPS points at each timestamp. After that, we compare the accuracy of query processing in the original dataset and simplified dataset with missing points interpolated.

4.5.3 Experimental Evaluation

The F-measure results of W-RQ are reported in Table 8. The dataset of Illinois is sampled at a very high rate and its GPS points are highly redundant. Thus, with the same compression ratio as the other datasets, it still preserves all the key information. For all three types of query operators, its accuracy among all algorithms is equal to 1. Similarly, the GeoLife dataset is more redundant than Taxi and Truck. The accuracies for most of the algorithms can also reach 100% in this dataset. In the following, we conduct the performance analysis and summarize the observations mainly using the results in Taxi and Truck datasets.

In the batch mode, the direction-aware methods work poorly and their simplified trajectories cannot well support popular queries in spatial-temporal database. Its F-measure in range query processing even drops to around 50% in the Truck dataset. This is because they put direction as the first-class citizen and miss plenty of key position and temporal information. The DP family (including **DP**, **DPhull** and **TD-TR**) shows comparable performance. Among them, **TD-TR** is slightly better in certain cases as it can capture the temporal dimension of trajectories. **MRPA** achieves superior performance among all the three operators and demonstrates better generality. Our explanation is that their distance measure ISSD is cumulative and defined on a local segment. It is more robust than PED or SED that is defined between the original location and a reference point.



(a) radius in W-RQ (b) window length in W-kNN

(c) k in W-kNN (d) dist threshold in W-TDJ

Figure 4: The accuracy of different query processing operators w.r.t. varying parameters.

In the online mode, the direction-aware methods (i.e., **Angular** and **Interval**) work significantly worse than the other methods as they are sensitive to direction variation and cannot well capture the location and temporal information. **DR**, **Threshold** and **Persistence** are early algorithms designed with linear complexity and simple objective. Even though they run fast, there is a noticeable performance margin between these two heuristic algorithms and the other competitors with higher computation complexity. The recent linear solutions such as **OPERB** and **ABQS** are improved with more complex processing logic, but still cannot achieve superior performance. This observation implies that with the ever increasing computing power in the sensors, it may not be preferable to design algorithms with linear complexity. When CPU and memory are not hard constraints, it is suggested to develop simplification algorithms that can trade these resources for better compression quality. **STTrace** and the **SQUISH** family share similar processing logic, by evoking the one with the minimum SED error or the smallest priority score when a new point arrives. These algorithms achieve comparable accuracies. **SQUISH-E(μ)** works slightly better than **SQUISH-E(λ)** and **SQUISH** because its objective is to minimize the number of simplified points in the meanwhile keeping the error below the threshold. With the same compression ratio, it allows a smaller error threshold. **DOTS** is the algorithm

Table 8: Performance of query processing operators on the reduced trajectory databases.

Batch Methods												
	F-measure of range query			Accuracy of kNN query			F-measure of join query			Accuracy of clustering		
	Taxi	Truck	GeoLife	Taxi	Truck	GeoLife	Taxi	Truck	GeoLife	Taxi	Truck	GeoLife
DPhull	0.971	0.978	1	0.440	0.863	0.996	0.674	0.862	1	1	0.999	1
TD-TR	0.973	0.999	1	0.432	0.969	0.997	0.680	0.957	1	1	0.999	1
MRPA	0.981	0.998	1	0.602	0.968	0.992	0.886	0.973	1	1	0.999	1
SP	0.658	0.457	0.795	0.36	0.747	0.980	0.512	0.777	0.995	1	0.999	0.997
Intersect	0.649	0.454	0.829	0.483	0.754	0.841	0.626	0.807	0.962	1	0.99	0.997
Error-Search	0.695	0.482	0.957	0.391	0.770	0.963	0.577	0.799	0.991	1	0.997	0.997
Span-Search	0.617	0.452	0.920	0.416	0.76	0.961	0.585	0.832	0.986	1	0.994	0.997
Online Methods												
	F-measure of range query			Accuracy of kNN query			F-measure of join query			Accuracy of cluster		
	Taxi	Truck	GeoLife	Taxi	Truck	GeoLife	Taxi	Truck	GeoLife	Taxi	Truck	GeoLife
Uniform	0.763	0.709	0.966	0.459	0.899	0.965	0.684	0.910	0.954	1	0.999	0.997
OPW	0.959	0.970	1	0.402	0.840	0.995	0.635	0.800	0.990	1	0.996	0.999
OPW-TR	0.980	0.984	1	0.580	0.923	0.992	0.839	0.893	1	1	0.999	0.999
DR	0.804	0.955	1	0.391	0.858	0.993	0.679	0.825	1	1	0.999	0.999
Threshold	0.866	0.612	0.929	0.270	0.715	0.952	0.514	0.696	0.982	1	0.999	0.997
STTrace	0.973	0.959	0.994	0.590	0.900	0.980	0.715	0.940	1	1	0.996	0.997
SQUISH	0.933	0.926	0.987	0.451	0.897	0.976	0.804	0.869	0.995	1	0.996	0.997
CDR	0.969	0.999	1	0.393	0.911	0.991	0.742	0.939	1	1	0.999	0.999
SQUISH-E(λ)	0.925	0.893	0.986	0.513	0.915	0.983	0.876	0.902	1	1	0.999	0.997
SQUISH-E(μ)	0.979	0.995	1	0.583	0.957	0.996	0.866	0.955	1	1	0.997	0.999
BQS	0.955	0.973	1	0.414	0.851	0.997	0.696	0.794	1	1	0.996	1
FBQS	0.955	0.973	1	0.428	0.864	0.996	0.708	0.818	1	1	0.999	1
Angular	0.744	0.450	0.812	0.334	0.686	0.944	0.303	0.707	0.995	0.480	0.574	0.794
Interval	0.620	0.494	0.871	0.278	0.650	0.936	0.255	0.728	0.995	0.429	0.496	0.798
OPERB	0.933	0.954	1	0.366	0.745	0.992	0.464	0.764	0.995	0.526	0.633	0.962
DOTS	0.982	0.998	1	0.660	0.933	0.996	0.618	0.917	1	1	0.997	1
ABQS	0.954	0.973	1	0.411	0.864	0.996	0.699	0.818	1	0.995	0.999	1
Persistence-online	0.913	0.905	0.979	0.338	0.931	0.993	0.554	0.882	0.995	1	0.996	0.994

that obtains dominating superiority among the three query processing operators. It is essentially an online version of **MRPA** and it further verifies that ISSD is a robust distance measure in both online and batch simplification modes. As to the accuracy of clustering, most of the algorithms can still preserve the trajectory similarity with compression ratio set to 10. All the batch algorithms achieve perfect clustering results. Only three algorithms in the online mode, including **Angular**, **Interval** and **OPERB**, perform poorly in the task of clustering. The reason could be that the similarity measure used in trajectory clustering takes into account perpendicular distance, parallel distance and angle distance, and is robust to different error metrics used in the simplification algorithms. The three algorithms with poor performance conduct only one-pass scan on the datasets and may miss sketch information that are important for clustering. Note that the result patterns are not sensitive to the number of clusters as we also conducted experiments on varying number of clusters (up to 3000) and obtained similar findings.

We also evaluate the query processing performance with varying parameters in the Taxi dataset. For each operator, we pick the three best algorithms in the batch mode and five best algorithms in the online mode. Figure 4 depicts the trending patterns with varying parameters in different types of operators. We can see that the F1-measure in W-RQ (or W-TDJ) increases with larger radius (or distance threshold). MRPA consistently outperforms the remaining batch algorithms in various scenarios. In the online mode, DOTS also demonstrates stable superiority against other online algorithms.

Finally, we conduct a correlation analysis among the query processing operators and error metrics. We use Pearson product-moment correlation [40], i.e, $\rho_{X,Y} = \frac{\sum(X-\bar{X})(Y-\bar{Y})}{\sqrt{\sum(X-\bar{X})^2 \sum(Y-\bar{Y})^2}}$. The error-metric vector contains the opposite number of error values and the query-metric vector contains the accuracy values. As shown in Figure 5, SED is closely related to PED as they both measure the error in the Euclidean space and SED is essentially an extension of PED. We also observe that SED is also positively correlated to Speed and W-kNN because Speed and the EDR measure in W-kNN take into account both position and temporal information. The direction-aware metric is negatively correlated with most of the metrics. The metric with the highest correlation with direction-metric is the accuracy of trajectory clustering. In fact, trajectory clustering does not show clear preference on certain simplification algorithms.

5. CONCLUSION AND FUTURE DIRECTION

In this paper, we conduct so far the most comprehensive evaluation on trajectory simplification with 25 algorithms and 5 GPS datasets. In addition, we examine the data usability as an alternative performance indicator for compression quality. Three types of popular spatio-temporal query operators are evaluated on the reduced database. The key observations worth noting are summarized as following:

1. The direction-aware trajectory simplification algorithms work poorly in practice. Even though they

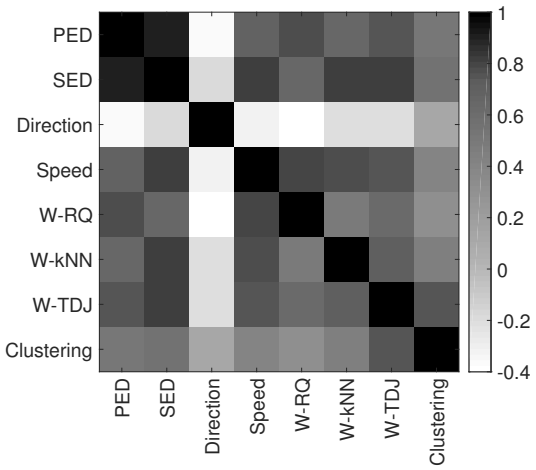


Figure 5: Correlation analysis of the error metrics.

are theoretically shown to preserve the position information as well, they cannot kill two birds with one stone. The derived errors under multiple distance metrics are dramatic. Moreover, their accuracies in supporting spatio-temporal operators in the reduced database are also disappointing. Thus, this line of algorithms are recommended only in a specific application where direction is the most important information to preserve.

2. The distance metric used for optimization plays a crucial role in determining the compression quality. ISSD is a cumulative distance measure on top of SED and the two algorithms **MRPA** and **DOTS** designed for ISSD perform persistently well among various testing scenarios. The algorithms with SED metric work noticeably better than those with PED in most cases, as they further consider the temporal dimension of trajectories. The algorithms designed for minimizing direction errors perform the worst.

3. Unless in an application with very special requirements, **MRPA** and **DOTS** are two algorithm recommended for batch and online modes, respectively. This is because they achieve the best performance in multiple error metrics and are effective in supporting spatio-temporal query processing in the reduced trajectory database.

We also present several future directions in trajectory simplification that may be worthy of exploration:

1. Nowadays, the cost of hardware has been decreasing. When the CPU and memory resources are not hard constraints, it is suggested to trade the computation time and memory space for higher compression quality. For instances, **MRPA** and **DOTS** apply a more complex distance metric, which requires more computation time. They are essentially trading calculation cost for more effective simplification.

2. Designing an effective distance metric is crucial for compression performance. The problem of trajectory simplification can be viewed as an instance of similarity measure learning and we can explore the possibilities of learning a new error metric that is suitable for a concrete application.

3. The current trajectory simplification algorithms only leverage the spatial and temporal redundancy within a particular trajectory. It may be interesting to explore the inter-trajectory redundancy to further improve the compression quality.

6. ACKNOWLEDGEMENT

This work is supported in part by the National Natural Science Foundation of China under grants No. 61602087, 61632007, 61702320, 61602488 and 61632016, the 111 Project No. B17008, the Fundamental Research Funds for the Central Universities under grants No. ZYGX2016J080, ZYGX2014Z007, the Talent Programme of Renmin University of China, the Tencent Social Ads Rhino-Bird Focused Research Grant and the Shanghai Municipal Education Commission Funds of Young Teacher Training Program No.ZZSDJ17021.

7. REFERENCES

[1] T. Ahmed, T. B. Pedersen, and H. Lu. Finding dense locations in symbolic indoor tracking data: modeling, indexing, and processing. *GeoInformatica*, 21(1):119–150, 2017.

[2] R. Bellman. On the approximation of curves by line segments using dynamic programming. *Commun. ACM*, 4(6):284–, June 1961.

[3] H. Cao and O. Wolfson. Nonmaterialized motion information in transport networks. In *ICDT*, pages 173–188, 2005.

[4] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *VLDB J.*, 15(3):211–228, 2006.

[5] W. Cao and Y. Li. Dots: An online and near-optimal trajectory simplification algorithm. *Journal of Systems and Software*, 126(Supplement C):34 – 44, 2017.

[6] L. Chen and R. T. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[7] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[8] M. Chen, M. Xu, and P. Fränti. A fast $o(n)$ multiresolution polygonal approximation algorithm for GPS trajectory simplification. *IEEE Trans. Image Processing*, 21(5):2770–2785, 2012.

[9] Y. Chen and J. M. Patel. Design and evaluation of trajectory join algorithms. In *GIS*, pages 266–275, 2009.

[10] P. Cudré-Mauroux, E. Wu, and S. Madden. Trajstore: An adaptive storage system for very large trajectory data sets. In *ICDE*, pages 109–120, 2010.

[11] D. H. Douglas and T. 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, 1973.

[12] L. Duan, T. Pang, J. Nummenmaa, J. Zuo, P. Zhang, and C. Tang. Bus-olap: A data management model for non-on-time events query over bus journey data. *Data Science and Engineering*, 3(1):52–67, Mar 2018.

[13] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu. SEAL: spatio-textual similarity search. *PVLDB*, 5(9):824–835, 2012.

[14] Q. Fan, D. Zhang, H. Wu, and K. Tan. A general and parallel platform for mining co-movement patterns over large-scale trajectories. *PVLDB*, 10(4):313–324, 2016.

[15] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis. Nearest neighbor search on moving object trajectories. In *SSTD*, pages 328–345, 2005.

[16] J. Hershberger and J. Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. Technical report, Vancouver, BC, Canada, Canada, 1992.

[17] G. Hu, J. Shao, F. Liu, Y. Wang, and H. T. Shen. If-matching: Towards accurate map-matching with information fusion. *IEEE Trans. Knowl. Data Eng.*, 29(1):114–127, 2017.

[18] P. Katsikouli, R. Sarkar, and J. Gao. Persistence based online signal and trajectory simplification for mobile devices. In *GIS*, pages 371–380, 2014.

[19] B. Ke, J. Shao, and D. Zhang. An efficient online approach for direction-preserving trajectory simplification with interval bounds. In *MDM*, 2017.

[20] B. Ke, J. Shao, Y. Zhang, D. Zhang, and Y. Yang. An online approach for direction-based trajectory compression with error bound guarantee. In *APWeb*, pages 79–91, 2016.

[21] G. Kellaris, N. Pelekis, and Y. Theodoridis. Map-matched trajectory compression. *Journal of Systems and Software*, 86(6):1566–1579, 2013.

[22] E. J. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.

[23] R. Lange, F. Dürr, and K. Rothermel. Efficient real-time trajectory tracking. *VLDB J.*, 20(5):671–694, 2011.

[24] J. Lee, J. Han, and K. Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD*, pages 593–604, 2007.

[25] C. Lin, C. Hung, and P. Lei. A velocity-preserving trajectory simplification approach. In *TAAI*, pages 58–65, 2016.

[26] X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai. One-pass error bounded trajectory simplification. *PVLDB*, 10(7):841–852, 2017.

[27] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak. Bounded quadrant system: Error-bounded trajectory compression on the go. In *ICDE*, pages 987–998, 2015.

[28] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, J. Lee, and R. Jurdak. A novel framework for online amnesic trajectory compression in resource-constrained environments. *IEEE Trans. Knowl. Data Eng.*, 28(11):2827–2841, 2016.

[29] K. Liu, Y. Li, J. Dai, S. Shang, and K. Zheng. Compressing large scale urban trajectory data. In *CloudDP*, pages 3:1–3:6, 2014.

[30] C. Long, R. C. Wong, and H. V. Jagadish. Direction-preserving trajectory simplification. *PVLDB*, 6(10):949–960, 2013.

[31] C. Long, R. C. Wong, and H. V. Jagadish. Trajectory simplification: On minimizing the direction-based error. *PVLDB*, 8(1):49–60, 2014.

[32] N. Meratnia and R. A. de By. Spatiotemporal compression techniques for moving point objects. In *EDBT*, pages 765–782, 2004.

[33] J. Muckell, J. Hwang, C. T. Lawson, and S. S. Ravi. Algorithms for compressing GPS trajectory data: an empirical evaluation. In *GIS*, pages 402–405, 2010.

[34] J. Muckell, J. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. S. Ravi. SQUISH: an online approach for GPS trajectory compression. In *COM.Geo*, pages 13:1–13:8, 2011.

[35] J. Muckell, P. W. Olsen, J. Hwang, C. T. Lawson, and S. S. Ravi. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica*, 18(3):435–460, 2014.

[36] A. Nibali and Z. He. Trajic: An effective compression system for trajectory data. *IEEE Trans. Knowl. Data Eng.*, 27(11):3138–3151, 2015.

[37] M. Potamias, K. Patroumpas, and T. K. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *SSDBM*, pages 275–284, 2006.

[38] R. Song, W. Sun, B. Zheng, and Y. Zheng. PRESS: A novel framework of trajectory compression in road networks. *PVLDB*, 7(9):661–672, 2014.

[39] G. Trajcevski, H. Cao, P. Scheuermann, O. Wolfson, and D. Vaccaro. On-line data reduction and the quality of history in moving objects databases. In *MobiDE*, pages 19–26, 2006.

[40] E. Turunen. Using guha data mining method in analyzing road traffic accidents occurred in the years 2004–2008 in finland. *Data Science and Engineering*, 2(3):224–231, Sep 2017.

[41] R. van Hunnik. Extensive comparison of trajectory simplification algorithms. In *Master Thesis*, 2017.

[42] M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.

[43] Y. Wang, D. Zhang, Q. Liu, F. Shen, and L. H. Lee. Towards enhancing the last-mile delivery: An effective crowd-tasking model with scalable solutions. *Transportation Research Part E: Logistics and Transportation Review*, 93:279 – 293, 2016.

[44] D. Yang, J. Guo, Z.-J. Wang, Y. Wang, J. Zhang, L. Hu, J. Yin, and J. Cao. Fastpm: An approach to pattern matching via distributed stream processing. *Information Sciences*, 2018.

[45] D. Zhang, D. Yang, Y. Wang, K. Tan, J. Cao, and H. T. Shen. Distributed shortest path query processing on dynamic road networks. *VLDB J.*, 26(3):399–419, 2017.

[46] R. Zhong, J. Fan, G. Li, K. Tan, and L. Zhou. Location-aware instant search. In *CIKM*, pages 385–394, 2012.