Geoinformatica
DOI 10.1007/s10707-013-0184-0

Compression of trajectory data: a comprehensive evaluation and new approach

Jonathan Muckell · Paul W. Olsen Jr. · Jeong-Hyon Hwang · Catherine T. Lawson · S. S. Ravi

Received: 13 July 2012 / Revised: 19 May 2013 / Accepted: 14 June 2013

Abstract GPS-equipped mobile devices such as smart phones and in-car navigation units are collecting enormous amounts of spatial and temporal information that traces a moving object's path. The exponential increase in the amount of such trajectory data has caused three major problems. First, transmission of large amounts of data is expensive and time-consuming. Second, queries on large amounts of trajectory data require computationally expensive operations to extract useful patterns and information. Third, GPS trajectories often contain large amounts of redundant data that waste storage and cause increased disk I/O time. These issues can be addressed by algorithms that reduce the size of trajectory data. A key requirement for these algorithms is to minimize the loss of information essential to location-based applications. This paper presents a new compression method called SQUISH-E (Spatial QUalIty Simplification Heuristic - Extended) that provides improved runtime performance and usability. A comprehensive comparison of SQUISH-E with

J. Muckell (✉)
Department of Informatics, College of Computing and Information,
University at Albany – State University of New York, Albany, NY 12222, USA
e-mail: jonmuckell@gmail.com

P. W. Olsen Jr. · J.-H. Hwang · S. S. Ravi
Department of Computer Science, College of Computing and Information,
University at Albany – State University of New York, Albany, NY 12222, USA

P. W. Olsen Jr.
e-mail: protsoph@gmail.com

J.-H. Hwang
e-mail: jhh@cs.albany.edu

S. S. Ravi
e-mail: ravi@cs.albany.edu

C. T. Lawson
Department of Geography and Planning and College of Computing and Information,
University at Albany – State University of New York, Albany, NY 12222, USA
e-mail: lawsonc@albany.edu

Published online: 04 July 2013

other algorithms is carried out through an empirical study across three types of real-world datasets and a variety of error metrics.

Keywords Trajectories · Compression · GIS

1 Introduction

In recent years, the number of GPS-enabled devices sold has drastically increased, following an impressive exponential trend. Canalys, an information technology firm that studies market patterns, reported a 116 % increase in the number of GPS units sold between 2006 and 2007 [1]. Location-based services and applications built from GPS-equipped mobile devices represent a rapidly expanding consumer market. In 2009, there was an estimated 27 million GPS-equipped smart phones sold, bringing the world-wide GPS user-base to at least 68 million in the consumer market alone [2]. These devices have the ability to generate, store and transmit trajectory data. A *trajectory* is defined as a stream of 3-tuple records consisting of the position (latitude, longitude), along with the temporal information (when the moving object was at the location).

Three major problems currently exist in location-based applications that use trajectory data. First, storing the sheer volume of trajectory data can quickly overwhelm available data storage space. For instance, if data is collected without compression at 10 s intervals, 1 Gb of storage capacity is required to store just over 4,000 objects for a single day [3]. For this reason, efficiently storing and querying GPS trajectories is an area of active research [3–6]. Second, the cost of sending a large amount of trajectory data over cellular or satellite networks can be expensive, typically ranging from $5 to $7 per megabyte [7]. Thus, for example, tracking a fleet of 4,000 vehicles for a single day would incur a cost of $5,000 to $7,000, or approximately $1,800,000 to $2,500,000 annually. Third, as the trajectory data size gets larger, it becomes more difficult to detect useful patterns from the data. Reducing the size of the trajectory data has the potential to accelerate the mining of trajectory patterns [8].

This paper extends two of our previous conference publications. The first publication [9] empirically compared seven existing algorithms for compressing trajectories. This work provided a discussion on the strengths and weaknesses of these algorithms and provided guidelines for selecting application-specific compression algorithms. A follow-up publication [10] presented a new trajectory compression algorithm. This algorithm, called SQUISH (Spatial QUalIty Simplification Heuristic), uses a priority queue where the priority of each point is defined as an estimate of the error that the removal of that point would introduce. SQUISH compresses each trajectory by removing points of the lowest priority from the priority queue until it achieves the target compression ratio. This algorithm is fast and tends to introduce small errors during compression. However, it cannot compress trajectories while ensuring that the resulting error is within a user-specified bound. It also exhibits relatively large errors when the compression ratio is high.

This paper presents a new version of SQUISH, called SQUISH-E (Spatial QUalIty Simplification Heuristic - Extended), that overcomes the limitations mentioned above. In contrast to its predecessor, this new algorithm provides provable

guarantees on the error caused by compression. Given a trajectory T and parameters λ and μ, this algorithm first compresses T while striving to minimize the error due to compression and ensuring the compression ratio of λ. It then further compresses T as long as this compression will not increase errors beyond μ. In this way, the new SQUISH-E algorithm allows users to control compression with respect to both compression ratio and error, sharply contrasting with previous compression algorithms. In particular, setting μ to 0 causes this algorithm to minimize error while achieving the compression ratio of λ. When λ is 1, this algorithm maximizes compression ratio while keeping errors under μ.

Other contributions of this paper include a comprehensive performance evaluation and comparison across three large datasets representing different travel modes (e.g, bus, urban commuter, and multi-modal travel which involves walking, cycling and rail). We analyzed the influence of travel modes on trajectory data particularly in terms of changes in direction, speed and acceleration. We also measured the impact of such changes on the degree of errors that each compression algorithm introduces (e.g., frequent, often unpredictable changes in movement make it difficult to compress trajectories with high accuracy). In our previous work [9], all of the compression algorithms were implemented in Matlab, which had limitations in code efficiency. We reimplemented these algorithms in Java and evaluated them with a focus on their unique trade-offs between compression speed and accuracy.

In this paper, we make the following contributions:

- We present a new algorithm, SQUISH-E, that compresses trajectories with provable guarantees on errors. This algorithm has the flexibility of tuning compression with respect to compression ratio and error.
- We provide formal proofs for the correctness of SQUISH-E.
- We present a comprehensive evaluation of trajectory compression algorithms using datasets that represent various data profiles (urban commuter, bus and multi-modal).
- We provide guidelines for choosing a compression algorithm that is best suited to the characteristics of trajectories as well as the organizational and technical requirements.

The remainder of this paper is organized as follows. Section 2 describes previous work for compressing trajectories, along with metrics for comparing trajectory compression algorithms. In Section 3, our new SQUISH-E algorithm is described in detail. An empirical evaluation of trajectory compression algorithms is provided in Section 4. The paper concludes with recommendations for future work in Section 5.

2 Background

Compression algorithms can be classified into two categories, namely lossless and lossy compression. Lossless compression enables exact reconstruction of the original data with no information loss. In contrast, lossy compression introduces inaccuracies when compared to the original data. The primary advantage of lossy compression is that it can often drastically reduce storage requirements while maintaining an acceptable degree of error. Due to this benefit, this paper focuses on lossy

compression of trajectory data. This section presents a comprehensive survey of metrics for comparing lossy trajectory compression algorithms (Section 2.1) and a summary of these algorithms (Section 2.2).

2.1 Metrics

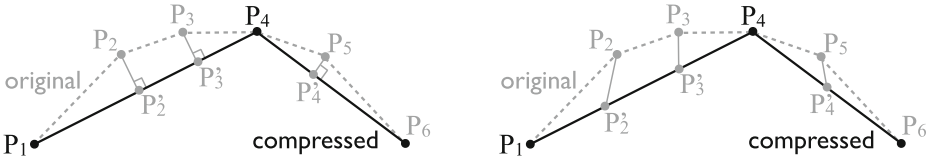
A trajectory T of length n consists of points $P_i(x_i, y_i, t_i)$ for $i \in \{1, 2, \cdots, n\}$, where x_i and y_i are the longitude and latitude of a moving object at time t_i. Trajectory compression algorithms described in Section 2.2 can compress trajectory T into another trajectory T' which contains a subset of points from T (i.e., $P_j(x_j, y_j, t_j)$ for $j \in M \subset \{1, 2, \cdots, n\}$). Metrics for expressing the difference between a trajectory and its compressed representation are explained in Section 2.1.1 and metrics for comparing the performance of trajectory compression algorithms are summarized in Section 2.1.2.

2.1.1 Accuracy metrics

Given a trajectory T and its compressed representation T', the *spatial error* of T' with respect to a point P_i in T is defined as the distance between $P_i(x_i, y_i, t_i)$ and its estimation $P'_i(x'_i, y'_i, t_i)$. If T' contains P_i, then P'_i is P_i (e.g., $P'_1 = P_1$ and $P'_4 = P_4$ in Fig. 1a where a trajectory containing $P_1, P_2, \cdots, P_6$ is approximated using only P_1, P_4 and P_6). Otherwise, P'_i is defined as the closest point to P_i along the line between $pred_{T'}(P_i)$ and $succ_{T'}(P_i)$, where $pred_{T'}(P_i)$ and $succ_{T'}(P_i)$ denote P_i's closest predecessor and successor among the points in T'. In Fig. 1a, $pred_{T'}(P_2) = P_1$ and $succ_{T'}(P_2) = P_4$. Therefore, the spatial error of T' with respect to P_2 is the perpendicular distance from P_2 to line $\overline{P_1 P_4}$.

Spatial error does not incorporate temporal data. *Synchronized Euclidean Distance (SED)* overcomes this limitation [11]. As in the case of spatial error, the SED between point $P_i(x_i, y_i, t_i)$ and its estimation $P'_i(x'_i, y'_i, t_i)$, denoted as $d(P_i, P'_i)$, is defined as the distance between (x_i, y_i) and (x'_i, y'_i). However, x'_i and y'_i are obtained for time point t_i via linear interpolation between $pred_{T'}(P_i)$ and $succ_{T'}(P_i)$. For instance, if $t_i = i$ for $i \in \{1, 2, 3, 4\}$ in Fig. 1b, P'_2 and P'_3 correspond to the points that divide the line between P_1 and P_4 into three line segments of the same length.

In addition to SED, many applications require accurate preservation of *speed* and *heading* information. For example, law enforcement needs accurate speed information to derive speeding hot-spots [12]. Furthermore, traffic flow modeling



(a) Spatial Error (b) Synchronized Euclidean Distance

Fig. 1 Compression accuracy metrics: Spatial error and SED are illustrated using a trajectory consisting of $P_1, P_2, \cdots, P_6$ and its compressed representation which contains P_1, P_4 and P_6

requires records about sharp changes in speed and heading to detect erratic behavior or disturbances in traffic flow [13]. Both speed and heading errors are determined in a way similar to SED, except that they capture the difference in speed and heading between the actual and estimated movements. Given points $P_i(x_i, y_i, t_i)$ and $P_{i+1}(x_{i+1}, y_{i+1}, t_{i+1})$, the speed at P_i is computed as $\frac{d(P_i, P_i+1)}{t_{i+1}-t_i}$. For estimations P'_i and P'_{i+1} of P_i and P_{i+1}, the speed at P'_i is similarly calculated. The speed error of T' with respect to P_i is defined as the difference between the speed at P_i and the estimated speed at P'_i.

Each of the metrics mentioned above expresses the error of a compressed representation T' with respect to a point in the original trajectory T. The overall error of T' with respect to T can be defined as an aggregate value (e.g., average, maximum) over the error values computed with respect to all of the points in T. For example, the maximum SED of T' is defined as $\max\{SED_{T'}(P_i) : P_i \in T\}$ where $SED_{T'}(P_i)$ denotes the SED of T' with respect to P_i.

2.1.2 Performance metrics

In addition to the *accuracy* metrics mentioned in Section 2.1.1, there are other types of metrics which are used for comparing the *performance* of trajectory compression algorithms. One metric, called *compression time*, refers to the amount of time that it takes to compress a trajectory. Another metric, *compression ratio*, is defined as the size of the original trajectory divided by the size of the compressed representation of that trajectory. For example, a compression ratio of 50 indicates that 2 % of the original points remain in the compressed representation of the trajactory.

2.2 Trajectory compression algorithms

Various trajectory compression algorithms exist in the literature. Each offers a different tradeoff among compression time, compression ratio, and accuracy. This section summarizes these algorithms.

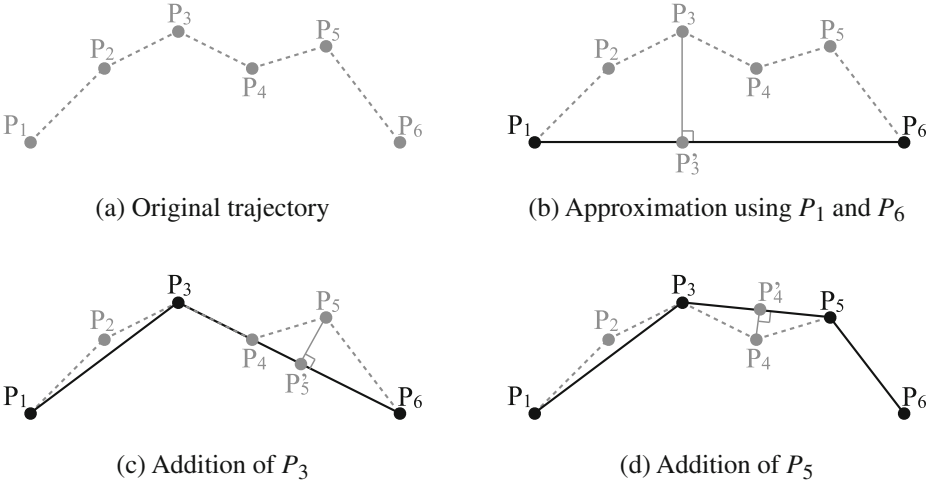
2.2.1 Uniform sampling

Given a trajectory T and the target compression ratio λ, Uniform Sampling downsamples T at fixed time intervals in a manner that achieves the compression ratio of λ. Uniform Sampling is fast, but often introduces large spatial and SED errors.

2.2.2 Douglas-Peucker

Given a trajectory T and a parameter Ψ, the Douglas-Peucker Algorithm [14] constructs a new trajectory T' by repeatedly adding points from T until the maximum spatial error of T' becomes smaller than Ψ. Figure 2 illustrates the operation of this algorithm. This algorithm initially approximates the original trajectory using the first and last points of the trajectory (Fig. 2b). Then, it repeats the process of selecting the point that causes the largest spatial error (e.g., P_3 in Fig. 2b and P_5 in Fig. 2c) and using that point to more accurately approximate the original trajectory. This process stops when the maximum spatial error (e.g., the distance from P_4 to $\overline{P_3 P_5}$ in Fig. 2d) is smaller than Ψ.

The worst-case running time of the original Douglas-Peucker algorithm is $O(n^2)$, where n is the number of original points. This running time can be improved to



(a) Original trajectory

(b) Approximation using P_1 and P_6

(c) Addition of P_3

(d) Addition of P_5

Fig. 2 Execution of Douglas-Peucker: Gray dots and dashed lines represent the original trajectory while black dots and solid lines represent the compressed representation of that trajectory. The compressed representation becomes a more accurate approximation of the original trajectory with the addition of a point whose absence caused the largest spatial error

O($n \log n$) using an approach that involves convex hulls [15]. A primary advantage of this algorithm is the guarantee that the resulting spatial error is less than the user-specified bound Ψ. A major drawback of Douglas-Peucker is that it ignores temporal data due to the use of spatial error. Douglas-Peucker also does not allow users to set the desired compression ratio.

2.2.3 Top-Down Time Ratio (TD-TR)

The Douglas-Peucker algorithm has the limitation of ignoring temporal data. Top-Down Time Ratio (TD-TR) [3] overcomes this limitation by using SED instead of spatial error. The worst-case running time of TD-TR is $O(n^2)$ since it extends the original Douglas-Peucker algorithm. The $O(n \log n)$ implementation of Douglas-Peucker [15] takes advantage of geometric properties that do not hold for SED. Therefore, it cannot be applied to TD-TR.

2.2.4 Opening Window algorithms

Similar to Douglas-Peucker, Opening Window algorithms [16] approximate each trajectory using an increasing number of points so that the resulting *spatial error* is smaller than a bound Ψ. A unique characteristic of Opening Window algorithms is that they slide a window over the points in the original trajectory. This window is initially anchored at the first point and gradually includes subsequent points until the distance from a point P_j in the window to the line segment between the first and last points in the window becomes larger than Ψ. Next, either point P_j is added to the compressed representation (Normal Opening Window Algorithm or NOWA) or the point right before P_j is added (Before Opening Window or BOPW). Such an added point is then used as the anchor of the next opening window. The above process is

repeated until the last point of the original trajectory is processed. The worst-case running time of Opening Window algorithms is $O(n^2)$ [16].

2.2.5 Opening Window Time Ratio (OPW-TR)

Opening Window Time Ratio (OPW-TR) [3] is an extension to Opening Window which uses *SED* instead of spatial error. Compared to Opening Window algorithms, OPW-TR has the benefit of taking temporal data into account. Just like Opening Window algorithms, OPW-TR has $O(n^2)$ worst-case running time.

2.2.6 Dead Reckoning

Dead Reckoning [17] stores the location of the first point P_1 and the velocity at P_1 in the compressed representation. It then skips every subsequent point P_i $(i > 1)$ whose location can be estimated from the information about P_1 within the SED of μ. If P_j is the first point whose location cannot be estimated as above, both the location of P_j and the speed at P_j are stored in the compressed representation, which are used for estimating the location of each point after P_j. This process is repeated until the input trajectory ends.

The computational complexity of Dead Reckoning is $O(n)$, where n is the number of points in the original trajectory. This complexity is due to the fact that it takes only $O(1)$ time to compare the actual and estimated locations of each point. The primary disadvantages of Dead Reckoning are that it tends to achieve lower compression ratios than other techniques (Section 4.3) and it does not allow users to set the target compression ratio.

2.2.7 Discussion

As summarized in Table 1, trajectory compression algorithms have different characteristics in terms of their guarantees on compression (e.g., Uniform Sampling achieves the compression ratio of λ while TD-TR limits SED under μ) and compression speed. Douglas-Peucker and TD-TR are offline algorithms that begin compression only after obtaining all of the points from the input trajectory. On the other hand, Uniform Sampling, Opening Window, OPW-TR, and Dead Reckoning are online algorithms which compress each trajectory while they retrieve points

Table 1 Summary of trajectory compression algorithms (n: trajectory size, λ: target compression ratio, Ψ: spatial error bound, μ: SED error bound)

Algorithm	Param.	Mode	Time complexity	SED (same compres. ratio)
Uniform Sampling	λ	online	$O(n)$	large
Douglas-Peucker	Ψ	offline	$O(n^2)$ [14], $O(n \log n)$ [15]	large
TD-TR	μ	offline	$O(n^2)$	small
Opening Window	Ψ	online	$O(n^2)$	large
OPW-TR	μ	online	$O(n^2)$	large
Dead Reckoning	μ	online	$O(n)$	large
SQUISH-E(λ)	λ	online/offline	$O(n \log \frac{n}{\lambda})$	small
SQUISH-E(μ)	μ	offline	$O(n \log n)$	small

from that trajectory. Such online algorithms have the advantages of supporting real-time applications and using a small amount of memory. Trajectory compression algorithms also introduce substantially different SED errors when their parameters are tuned so that they achieve the same compression ratio (Section 4).

Fast compression algorithms such as Uniform Sampling and Dead Reckoning tend to introduce large SED errors. Algorithms that use spatial error (i.e. Douglas-Peucker, Opening Window) also exhibit large SED errors since they ignore temporal data. Online compression algorithms tend to cause larger SED error than TD-TR. However, TD-TR incurs high computational overhead (i.e., low compression speed) since it recomputes the SED of the current compressed representation with respect to multiple points whenever it adds a point into the compressed representation.

The above shortcomings of the previous algorithms motivated the development of our SQUISH-E algorithm which is further described in Section 3. This algorithm supports online compression in that it starts removing points before it reaches the end of the input trajectory. It also has one characteristic of an offline algorithm in that it produces the final compressed representation only after it has accessed all of the points in the trajectory. Compared to TD-TR, SQUISH-E achieves competitive accuracy while incurring substantially lower computational and space overhead (Section 4).

The algorithms examined in this study approximate each trajectory using a subset of points from that trajectory. Among such point-selection algorithms, STTrace [11] and Bellman's Algorithm [18] are not included in this study because of their significantly higher error rates and computational costs compared to other algorithms [19]. Another point-selection algorithm, explored by Feldman et al. [20], uses *coresets* for approximating a large collection of sensor data. Given a large set of points S, a coreset C is a small subset S such that C can be used to approximate S [21]. For practical problems where the number of points is very large, the corresponding algorithms that provide good approximations tend to be computationally expensive [20].

In contrast to point-selection approaches, semantic compression [22] enables extreme compression by storing only the points that represent key events such as transport stops. Semantic compression is effective for applications that have fixed requirements. On the other hand, point-selection approaches have wider applicability since they use error metrics that consider only spatial and temporal data rather than other application-specific information. There has also been research on lossless compression of trajectory data. One such study by Lin et al. [23] uses inter-frame encoding. Lossless compression techniques in general achieve a lower compression ratio compared to lossy compression techniques. Lossy and lossless techniques can also be used in tandem by applying lossy compression followed by lossless compression to each trajectory.

Additional lossy compression algorithms and data structures have been proposed. One approach by Kaul et al. [24] approximates trajectories using polynomial prediction. A key finding in this work is that linear prediction which relies on the two most recent GPS updates outperforms more complex polynomial prediction techniques. This approach is not as accurate as Douglas-Peucker and conceptually similar to Dead Reckoning which also uses linear prediction. Potamias et al. proposed a data structure, called AM-Tree [25, 26], which stores the most recent information at a higher rate. This data structure is based on the assumption that the value of information decays over time.

3 Spatial QUalIty Simplification Heuristic - Extended (SQUISH-E)

A key contribution of our research is a novel approach called SQUISH-E for compressing trajectories. Given λ, the target compression ratio, our previous SQUISH algorithm [10] compresses a trajectory of length n into a trajectory of length at most n/λ. This algorithm enables highly accurate approximation of trajectories in a substantially shorter time than other techniques [10]. However, it lacks the capability of compressing trajectories while ensuring that SED error is within a user-specified bound. Our SQUISH-E algorithm overcomes this limitation. Algorithmic details and the correctness of this version are explained in Sections 3.1 and 3.2, respectively. Section 3.3 discusses the novelty and unique benefits of this version.

3.1 Algorithm description

Our new SQUISH-E algorithm requires a trajectory T to compress, and two additional parameters λ and μ. It compresses trajectory T while striving to minimize SED error and achieving the compression ratio of λ. Then, it further compresses T as long as this compression will not increase SED error beyond μ. Therefore, setting μ to 0 causes this algorithm to minimize SED error ensuring the compression ratio of λ. We refer to this case as SQUISH-E(λ). SQUISH-E(μ) denotes another case where λ is set to 1 and therefore SQUISH-E maximizes compression ratio while keeping SED error under μ.

The key idea of SQUISH-E is to use a priority queue Q, where the priority of each point is defined as an upper bound on the SED error that the removal of that point would introduce. Therefore, SQUISH-E can find and remove a point from Q that has the lowest priority (i.e., a point whose removal would increase SED error within the lowest bound) in $O(\log |Q|)$ time, where $|Q|$ denotes the number of points stored in Q. By removing points in this way, SQUISH-E can effectively limit the growth of SED error.

Algorithm 1 provides a detailed description of SQUISH-E. Table 2 summarizes the variables used in Algorithm 1. Figure 3 illustrates the operation of SQUISH-E using an example. In SQUISH-E, variable β stores the capacity of priority queue Q. The initial value of β is 4 (line 1 in Algorithm 1) so that Q can store the first four points (e.g., P_1, P_2, P_3 and P_4 in Fig. 3a). The value of β increases whenever $\frac{i}{\lambda} \geq \beta$, where i denotes the number of points retrieved so far from trajectory T (lines 3 and 4). Each point P_i from trajectory T (line 2) is initially assigned a priority of ∞ when it is inserted into Q (line 5). If P_i is not the first point (line 7), P_i is registered as the closest successor of its previous point P_{i-1} (line 8). P_{i-1} is also registered as the closest predecessor of P_i (line 9). Then, the priority of P_{i-1} is adjusted to the SED error that the removal of P_{i-1} would introduce (line 10). In Fig. 3a, the priority of P_2 is set to 0.5 since removing P_2 will cause the SED with respect to P_2 to be 0.5 (i.e., the distance between P_2 and P_2' in Fig. 3b). Further details of priority adjustment (Algorithm 3) are provided later in this section.

When Q is full (i.e., contains β points), SQUISH-E reduces Q by removing from Q a point that has the lowest priority (lines 11 and 12 in Algorithm 1). For example, in Fig. 3a, SQUISH-E would remove either P_2 or P_3 since they have the lowest priority of 0.5 among the four points in Q. Reducing Q as above ensures that Q keeps only $\beta - 1$ points out of the points seen so far in a manner that effectively limits the growth

Algorithm 1: SQUISH-E(T, λ, μ)

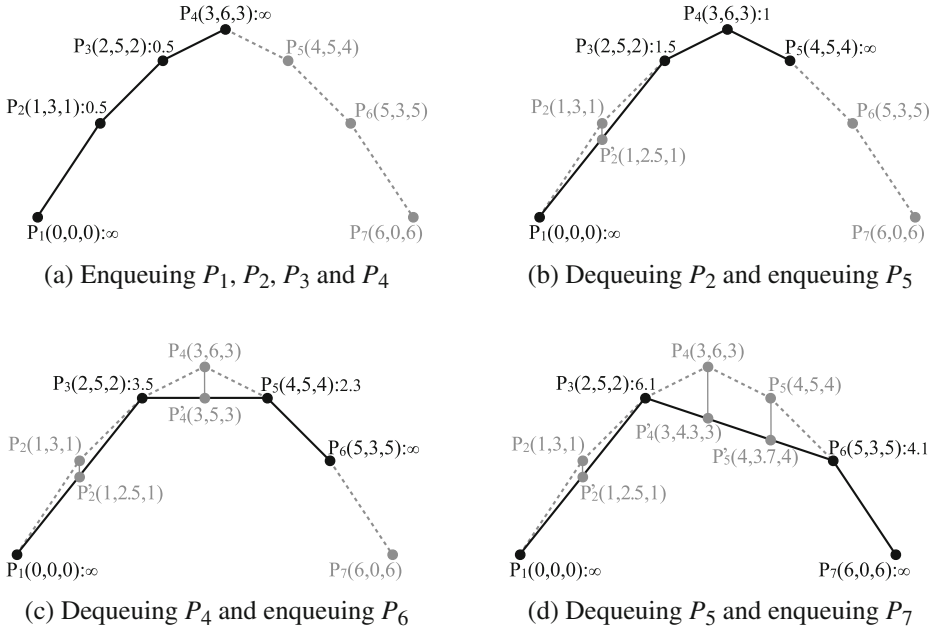
```
   input  : trajectory T, lower bound λ on compression ratio, upper bound μ on SED
   output : trajectory T'
1  β ← 4; // the initial capacity of Q is 4
2  for each point P_i ∈ T do
3      if i/λ ≥ β then
4          β ← β + 1; // increase the capacity of Q
5      set_priority(P_i, ∞, Q); // enqueue P_i with the priority of P_i being ∞
6      π[P_i] ← 0;
7      if i > 1 then // P_i is not the first point
8          succ[P_{i-1}] ← P_i; // register P_i as P_{i-1}'s closest successor
9          pred[P_i] ← P_{i-1}; // register P_{i-1} as P_i's closest predecessor
10         adjust_priority(P_{i-1}, Q, pred, succ, π); // Algorithm 3
11     if |Q| = β then // Q is full
12         reduce(Q, pred, succ, π); // Algorithm 2
13 p ← min_priority(Q); // find the lowest priority from Q
14 while p ≤ μ do // the lowest priority is not higher than μ
15     reduce(Q, pred, succ, π); // Algorithm 2
16     p ← min_priority(Q); // find the lowest priority from Q
17 return trajectory T' comprising the points in Q in the order reflected in the succ map;
```

of SED error. Once all of the points in T are processed, SQUISH-E finds the lowest priority p obtained from the priorities of the points remaining in Q (lines 13). If p is not higher than μ (i.e., removing a point which has the lowest priority will not increase SED error beyond μ), SQUISH-E reduces Q by removing a point which has the lowest priority (lines 14 and 15). This process of reducing Q is repeated until every point remaining in Q has a priority higher than μ.

Algorithm 2 describes the details of reducing priority queue Q. In this process, a point P_j which has the lowest priority is removed from Q (line 1). Next, the priority of P_j is used to update $\pi[succ[P_j]]$ and $\pi[pred[P_j]]$ (lines 2 and 3). For each point $P_k \in Q$, $\pi[P_k]$ stores the maximum of the priorities that the neighboring points of P_k had when they were removed from Q. If none of P_k's neighboring points has been removed, $\pi[P_k] = 0$ (line 6 in Algorithm 1). For example, in Fig. 3a, P_2 has the lowest priority of 0.5. Therefore, removing P_2 causes the values of $\pi[P_1]$ and $\pi[P_3]$ to change from 0 to 0.5. As explained below, maintaining $\pi[P_k]$ as above for each point $P_k \in Q$ allows SQUISH-E to derive an upper bound on the SED error that the removal of P_k would introduce (a formal proof is provided in Section 3.2).

Table 2 Variables used in Algorithm 1

Variable	Description
Q	priority queue
β	capacity of Q
pred	map storing, for each $P_i \in Q$, P_i's closest predecessor among the points in Q
succ	map storing, for each $P_i \in Q$, P_i's closest successor among the points in Q
π	map storing, for each $P_i \in Q$, the maximum of the priorities that the neighboring points of P_i had when they were removed from Q



(a) Enqueuing P_1, P_2, P_3 and P_4

(b) Dequeuing P_2 and enqueuing P_5

(c) Dequeuing P_4 and enqueuing P_6

(d) Dequeuing P_5 and enqueuing P_7

Fig. 3 Execution of SQUISH-E: Black dots represent the points kept in priority queue Q. The numbers within a pair of parentheses are the coordinate values of the corresponding point. The number after a pair of parentheses is the current priority of the corresponding point

In addition to updating $\pi[succ[P_j]]$ and $\pi[pred[P_j]]$ as above, SQUISH-E registers $succ[P_j]$ as the new closest successor of $pred[P_j]$ (lined 4 in Algorithm 2) and registers $pred[P_j]$ as the new closest predecessor of $succ[P_j]$ (line 5). In Fig. 3b, due to the removal of P_2, the closest successor of P_1 becomes P_3 (i.e., $succ[P_1] = P_3$) and the closest predecessor of P_3 becomes P_1 (i.e., $pred[P_3] = P_1$). Then, the priority of $pred[P_j]$ and that of $succ[P_j]$ are adjusted (lines 6 and 7 in Algorithm 2) using Algorithm 3. The reason for this adjustment is that the removal of P_j affects the SED error with respect to $pred[P_j]$ and $succ[P_j]$. For example, in Fig. 3a, the calculation of P_3's priority takes into account line segment $\overline{P_2P_4}$. After removing P_2 (Fig. 3b), that calculation needs to involve $\overline{P_1P_4}$ rather than $\overline{P_2P_4}$.

Algorithm 2: reduce$(Q, pred, succ, \pi)$

input : priority queue Q, maps $pred$, $succ$ and π (refer to Table 2)

1 $P_j \leftarrow$ remove_min(Q); // *point P_j which has the lowest priority is removed from Q*
2 $\pi[succ[P_j]] \leftarrow \max(priority(P_j), \pi[succ[P_j]])$; // *ensure that $\pi[succ[P_j]] \geq priority(P_j)$*
3 $\pi[pred[P_j]] \leftarrow \max(priority(P_j), \pi[pred[P_j]])$; // *ensure that $\pi[pred[P_j]] \geq priority(P_j)$*
4 $succ[pred[P_j]] \leftarrow succ[P_j]$; // *register $succ[P_j]$ as the closest successor of $pred[P_j]$*
5 $pred[succ[P_j]] \leftarrow pred[P_j]$; // *register $pred[P_j]$ as the closest predecessor of $succ[P_j]$*
6 adjust_priority$(pred[P_j], Q, pred, succ, \pi)$; // *Algorithm 3*
7 adjust_priority$(succ[P_j], Q, pred, succ, \pi)$; // *Algorithm 3*
8 remove the entry for P_j from $pred$, $succ$, and π; // *garbage collection*

Algorithm 3: adjust_priority$(P_j, Q, pred, succ, \pi)$

```
input      : point P_j, priority queue Q, maps pred, succ and π (refer to Table 2)
1 if pred[P_j] ≠ null and succ[P_j] ≠ null then
2     p ← π[P_j] + SED(P_j, pred[P_j], succ[P_j]);
3     set_priority(P_j, p, Q);
```

Algorithm 3 shows how SQUISH-E adjusts the priority of point P_j. If P_j is the first point (i.e., $pred[P_j] = \textbf{null}$) or the last point (i.e., $succ[P_j] = \textbf{null}$) among the points in Q (line 1), the priority of P_j remains at its initial value ∞ (line 5 in Algorithm 1). Otherwise, the priority of P_j is set to a new value p, which is the sum of $\pi[P_j]$ (i.e., the maximum of the priorities that P_j's neighboring points had when they were removed) and the SED between P_j and line segment $\overline{pred[P_j], succ[P_j]}$ (line 2 in Algorithm 3). For instance, when point P_2 whose priority is 0.5 in Fig. 3a is removed (Fig. 3b), $\pi[P_3]$ becomes 0.5 and $SED(P_3, P_1, P_4)$ becomes the distance between $P_3(2, 5, 2)$ and $P'_3(2, 4, 2)$, which is 1. For this reason, the priority of P_3 is set to $0.5 + 1 = 1.5$. After obtaining p as above, point P_j is first removed from Q and then inserted into Q with priority p (line 3). Section 3.2 provides a formal proof that p is an upper bound on the SED that the removal of P_j would introduce (Lemma 1).

3.2 Correctness of SQUISH-E

The λ parameter of SQUISH-E indicates a lower bound on the compression ratio. When λ is set to 1, the SQUISH-E algorithm must ensure that the actual SED error introduced during compression is no larger than μ. Theorems 1 and 2 verify the correctness of this algorithm:

Theorem 1 (Compression ratio) *Given a trajectory T, SQUISH-E produces a compressed representation T' of T so that $\frac{|T|}{|T'|} \geq \lambda$, where $|T|$ and $|T'|$ denote the lengths of T and T', respectively.*

Proof In Algorithm 1, for $\lambda \geq 1$, β is incremented whenever $\frac{i}{\lambda} \geq \beta$ (lines 3 and 4), meaning that $\frac{i}{\lambda} < \beta \leq \frac{i}{\lambda} + 1$ after this step. Whenever $|Q| = \beta$, a point is removed from Q (lines 11–12). More points may also be removed from Q after the end of T is reached (lines 13–16). Since T' consists of the points that ultimately remain in Q, $|T'| = |Q| \leq \beta - 1 \leq (\frac{i}{\lambda} + 1) - 1 = \frac{i}{\lambda} = \frac{|T|}{\lambda}$. Therefore, $\frac{|T|}{|T'|} \geq \frac{|T|}{\frac{|T|}{\lambda}} = \lambda$. □

Lemma 1 *After SQUISH-E reduces Q (lines 12 and 15 in Algorithm 1) using Algorithm 2, for each $P_j \in Q$ such that $1 < j < i$ and for each point P_k between $pred[P_j]$ and $succ[P_j]$ in the original trajectory T,*

$$priority(P_j) \geq SED(P_k, pred[P_j], succ[P_j]) \tag{1}$$

where $priority(P_j)$ denotes the priority of point P_j and $SED(P_k, pred[P_j], succ[P_j])$ denotes the SED between P_k and the line segment joining $pred[P_j]$ and $succ[P_j]$.

Proof We prove this lemma by induction as follows:

Base case Suppose that both P_{j-1} and P_{j+1} are in Q, meaning that $priority(P_j)$ did not change after it was adjusted to $\pi[P_j] + SED(P_j, pred[P_j], succ[P_j]) = 0 + SED(P_j, P_{j-1}, P_{j+1})$ (line 2 in Algorithm 3 and line 10 in Algorithm 1). In this case, Eq. 1 holds.

Induction step If P_{j-1} is not in Q, then let P_a be $pred[P_j]$, and P_b be the most recently removed point among the points between P_a and P_j in the original trajectory T (Fig. 4). Also, assume, as an induction hypothesis, that when P_b was removed, $priority(P_b) \geq SED(P_k, pred[P_b], succ[P_b])$ for each P_k between $pred[P_b] = P_a$ and $succ[P_b] = P_j$ in T (Fig. 4). Let P'_k denote the estimation of P_k in the case of SED (Section 2.1.1) along line segment $\overline{P_a P_j}$. Then, $\pi[P_j] = \pi[succ[P_b]] \geq priority(P_b)$ (line 2 in Algorithm 2) and $priority(P_b) \geq SED(P_k, pred[P_b], succ[P_b]) = d(P_k, P'_k)$. Therefore, $\pi[P_j] \geq d(P_k, P'_k)$ (i). Let P''_k and P''_j represent the estimation of P_k and that of P_j along $\overline{P_a P_c}$. Then, $SED(P_j, pred[P_j], succ[P_j]) = d(P_j, P''_j) > d(P'_k, P''_k)$ (ii) since the angle between $\overline{P_a P'_k}$ and $\overline{P_a P''_k}$ and the angle between $\overline{P_a P_j}$ and $\overline{P_a P''_j}$ are the same, P'_k and P''_k divide $\overline{P_a P_j}$ and $\overline{P_a P''_j}$, respectively, in the same proportion by the definition of SED (Section 2.1.1), and $\overline{P_a P_j}$ is longer than $\overline{P_a P'_k}$. At this point, due to line 2 in Algorithm 3, $priority(P_j) = \pi[P_j] + SED(P_j, pred[P_j], succ[P_j]) \geq d(P_k, P'_k) + d(P'_k, P''_k)$ by (i) and (ii). Furthermore, $d(P_k, P'_k) + d(P'_k, P''_k) \geq d(P_k, P''_k) = SED(P_k, pred[P_j], succ[P_j])$ by triangle inequality. Therefore, Eq. 1 holds for each P_k between $pred[P_j]$ and P_j in T (iii).

If P_{j+1} is not in Q, then it can also be proven as in the derivation of (iii) that Eq. 1 holds for each P_k between P_j and $succ[P_j]$ in T (iv). Since $priority(P_j) = \pi[P_j] + SED(P_j, pred[P_j], succ[P_j])$ (line 2 in Algorithm 3) and $\pi[P_j] \geq 0$ (line 6 in Algorithm 1 and lines 2 and 3 in Algorithm 3), Eq. 1 holds when P_k is P_j (v). By (iii), (iv), and (v), for each P_k between $pred[P_j]$ and $succ[P_j]$ in T, Eq. 1 holds. □

Theorem 2 (SED error) *Suppose that SQUISH-E compresses a trajectory T into T' with λ set to* 1. *Then, $SED_{T'}(P_i) \leq \mu$ for every $P_i \in T$, where $SED_{T'}(P_i)$ denotes the SED of T' with respect to point P_i.*

Proof For $\mu \geq 0$ and for each point $P_i \in T'$, $SED_{T'}(P_i) = 0 \leq \mu$ (Section 2.1.1). For an arbitrary point $P_i \in T - T'$, let P_j denote the most recently removed point such that P_i was between $pred[P_j]$ and $succ[P_j]$. Then, $SED_{T'}(P_i) =$

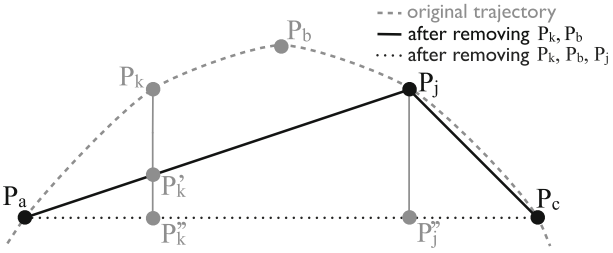


Fig. 4 Illustration of the induction step in the proof of Lemma 1

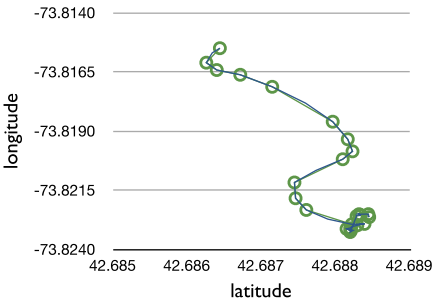
$SED(P_i, pred[P_j], succ[P_j]) \leq priority(P_j)$ when P_j was removed (Lemma 1) and $priority(P_j) \leq \mu$ (line 14 in Algorithm 1). Therefore, $SED_{T'}(P_i) \leq \mu$. □

3.3 Discussion

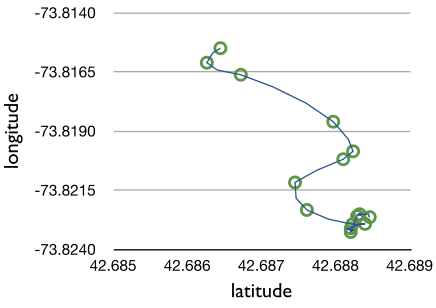
Compared to its predecessor [10], a major benefit of SQUISH-E is that it provides a provable guarantee on the SED error introduced during compression. It also has the benefit of supporting two special modes, namely SQUISH-E(λ) which minimizes SED error while achieving the compression ratio of λ and SQUISH-E(μ) which maximizes compression ratio while limiting SED error under μ. The previous SQUISH algorithm [10] is similar to SQUISH-E(λ) in nature, but different in that it lacks guarantees on SED error. Furthermore, SQUISH-E overcomes the previous version's limitation of ignoring compression opportunities once it achieves the compression ratio of λ. For instance, the previous SQUISH algorithm always compresses a trajectory T into another trajectory whose length is $|T|/\lambda$ even if the input trajectory T exhibits little variations in speed and heading. On the other hand, setting μ to a tolerable error value allows SQUISH-E to achieve a much higher compression ratio through additional removals which do not increase SED error beyond μ (Fig. 5). Furthermore, the previous SQUISH algorithm uses a fixed size priority queue which is constructed only after the length of the input trajectory and target compression ratio are given. In contrast, SQUISH-E dynamically increases the size of the priority queue and thus can be used for trajectories that are provided in the form of streaming data.

To achieve the above benefits, the details of SQUISH-E (particularly, the ways of updating variables and adjusting the priorities of points in Algorithms 1, 2, and 3) are substantially different from those of its previous version. These technical details (Section 3.1) and the formal proofs on the correctness of SQUISH-E (Section 3.2) are new contributions of this paper.

SQUISH-E has unique advantages over previous trajectory compression algorithms. First, SQUISH-E has the flexibility of controlling compression with respect to both compression ratio and SED error. Second, SQUISH-E enables fast compression.



(a) With λ = 5, SQUISH(λ) achieves a compression ratio of 5.



(b) With λ = 5, μ = 10 (meters), SQUISH-E(λ, μ) achieves a compression ratio higher than 5.

Fig. 5 Benefit of SQUISH-E(λ, μ)

The operation of removing a point with the lowest priority from Q takes $O(\log |Q|)$ time, where $|Q|$ is the number of points in Q. Given a trajectory T of length n, the running time and space overhead of SQUISH-E(λ) are $O(n \log \frac{n}{\lambda})$ and $O(\frac{n}{\lambda})$ since $|Q| \leq \frac{n}{\lambda}$. In the case of SQUISH-E(μ), the running time and space overhead are $O(n \log n)$ and $O(n)$, respectively, since $|Q| \leq n$. In addition to the above benefits, SQUISH-E achieves low SED error since it removes points whose removal will increase SED by the smallest possible amount. These characteristics of SQUISH-E are summarized in Table 1 and experimentally demonstrated in Section 4.

4 Evaluation

This section experimentally evaluates algorithms that compress trajectories by removing a subset of points from them. Our evaluation does not include STTrace [11] and Bellman's algorithm [18] which introduce larger errors despite longer running times compared to TD-TR [3] (refer to Section 2.2.7 for further details). The techniques that are evaluated in this section have unique benefits in balancing compression time and the degree of error (Table 1).

Section 4.1 describes the datasets used for evaluating trajectory compression algorithms. Sections 4.2 and 4.3 compare the trajectory compression algorithms in terms of compression time (Section 2.1.2) and accuracy/error metrics (Section 2.1.1), respectively. Section 4.4 presents application-specific recommendations for choosing trajectory compression algorithms.

4.1 Datasets

Table 3 summarizes the datasets used for evaluating trajectory compression algorithms. Each dataset represents a different transportation mode (bus, urban commuter, and multi-modal).

4.1.1 Microsoft GeoLife dataset

The Microsoft GeoLife [27, 28] dataset was obtained from 178 individuals over a period of two years (from April 2007 to August 2009). This dataset includes various transportation modes such as biking, walking, and rail. Approximately 91 % of the trajectories have a sampling rate of 1–5 s and others have higher sampling rates. Most of the data collection occurred around Beijing, China and a small number of trajectories were obtained in the United States and Europe.

To properly evaluate trajectory compression algorithms, we cleaned the GeoLife dataset. For example, we removed every part in the trajectories which was collected while the corresponding individual remained stationary for over 20 min. The reason for this removal is to prevent compression results from being skewed by unusually

Table 3 GPS trajectory datasets

Dataset	Location	Mode(s)	Trajectories	Points	Size	Sampling rate
GeoLife	Beijing, China	multi-modal	6,923	12,847	641 Mb	1–5 s
Bus	Albany, NY	bus	52	4,608	7 Mb	5 s
NYMTC	New York City	urban commuter	30	2,902	52 Mb	5 s

high data redundancy. We also split each trajectory whenever we detected a gap in recorded time which was at least 20 min long. Furthermore, points that exhibit unrealistic speed (e.g., 1000 km/hr) were removed from the trajectories. As a result, we obtained a total of 6,923 trajectories, each containing 12,847 points on average.

4.1.2 Albany bus transit dataset

The Bus dataset in Table 3 was obtained from buses traveling along four routes in Albany, New York, over a period of 12 weeks (October to December, 2009) [29]. This dataset consists of 52 trajectories with a combined total of roughly 240,000 spatio-temporal points collected every 5 s.

4.1.3 New York Metropolitan Transportation Council dataset

This dataset consists of trajectories collected at the sampling rate of 5 s by 24 volunteers from the New York Metropolitan Transportation Council (NYMTC) [30]. These trajectories reflect the movements of individuals commuting into New York City for work and returning home at the end of the day.

4.1.4 Discussion

Each dataset included in this study represents a unique data profile. Travel mode is a crucial predictor of such profiles. On average, as Table 4 shows, the GeoLife dataset has the largest changes in speed and direction, followed by the Bus and NYMTC datasets. The GeoLife dataset represents a multitude of different travel modes that occurred mainly in the complex urban environment of Beijing, China. Although filters were applied to clean the dataset, significant inaccuracies in the data are inevitable due to the urban canyon effect and obstructions that affect GPS readings. The Bus dataset represents noticeable changes in speed that took place as buses stopped due to traffic or the arrival at designated stops, or as GPS units inside a bus had an obstructed view of the sky. On the other hand, a large number of trajectories in the NYMTC dataset represent the urban commuter mode which typically involves a ride at a relatively constant speed, typically of a car or rail with smooth stops.

Table 5 shows how the characteristics of datasets affect the errors introduced during compression. The results in the table were obtained across all of the compression algorithms evaluated in this study. Specific breakdowns of the results per compression algorithm are provided and discussed in Section 4.3. Trajectories from the GeoLife dataset are by far the most difficult to compress with high accuracy due to the high average speed and large variations in speed. In contrast, trajectories from the NYMTC dataset were compressed with lowest SED, speed, and spatial errors. Table 5 shows that the ranking of accuracy is in general consistent across SED, speed,

Table 4 Statistics of datasets: Δ speed and Δ direction represent changes in speed and direction, respectively (speed unit: km/hr, direction unit: degrees)

Dataset	Avg(speed)	Std(speed)	Avg(Δ speed)	Std(Δ speed)	Std(Δ direction)
GeoLife	20.7	18.8	7.5	5.9	27.5
Bus	15.7	18.4	4.9	7.9	27.8
NYMTC	10.2	19.2	2.6	3.7	29.9

Table 5 Impact of Datasets on Compression Errors at a Fixed Compression Ratio of 10 (SED/spatial error unit: meters, speed unit: km/hr, heading unit: degrees)

Dataset	Svg(SED error)	Avg(spatial error)	Avg(speed error)	Avg(heading error)
GeoLife	75.0	66	24.1	25.7
Bus	33.1	50	6.28	42.3
NYMTC	12.7	43	2.39	52.5

and spatial errors. On the other hand, compressing trajectories from the NYMTC dataset resulted in large heading errors.

4.2 Evaluation based on compression times

Figure 6 depicts the actual execution times of each trajectory compression algorithm. This experiment used 71 trajectories, each containing a minimum of 20,000 points. To measure the effect of the trajectory size on compression time, we obtained sub-trajectories by taking the first 5,000, 10,000, 15,000, and 20,000 points from each of the original trajectories. To compare the algorithms on a fair basis, the compression ratio was set to 10 for all of these algorithms. However, most algorithms including Douglas-Peucker, Dead-Reckoning, Opening Window and SQUISH-E(μ) do not take a target compression ratio as an input parameter (Table 1). Therefore, we used an approach which repeatedly executed such an algorithm, modifying the error bound parameter as in binary search, until the desired compression ratio was achieved. To obtain reliable results on compression time despite short compression runs, we repeatedly ran each algorithm on the same trajectory until the accumulated execution time became greater than 1 s. Then, we averaged the execution times of these iterations to obtain the compression time for that combination of algorithm and trajectory. In this evaluation study, we used one core of a Xeon E5430 2.67 GHz CPU for each run.

In Fig. 6, the fastest algorithm is Uniform Sampling which compressed trajectories consisting of 20,000 points within 36 microseconds on average. For the same trajectories, the average execution time of TD-TR, the slowest among the eight algorithms, was 85 milliseconds, which was more than 2,000 times longer than that of Uniform Sampling. As Fig. 6 shows, TD-TR incurs significantly higher computational overhead than Douglas-Peucker although they are identical except for the error metrics that they use. The reason behind this difference is that SED error (TD-TR)

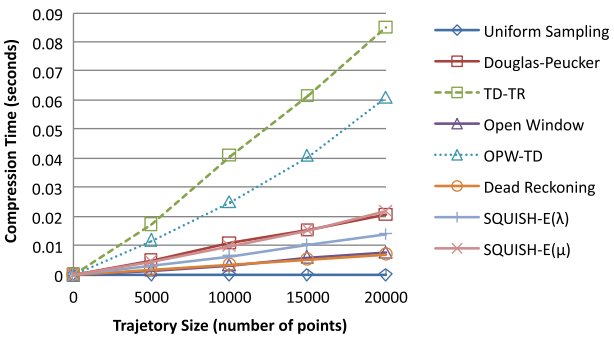


Fig. 6 Average compression time

requires more computation than spatial error (Douglas-Peucker) particularly due to the Haversine distance calculation between two points on a sphere [31]. For the same reason, OPW-TR is also substantially slower than Opening Window. The benefits of using SED error with respect to error metrics are experimentally demonstrated in Section 4.3. In our result, Dead Reckoning and Opening Window were the fastest algorithms after Uniform Sampling. Dead Reckoning has linear time overhead like Uniform Sampling (Table 1). In theory, Opening Window may exhibit quadratic time overhead (Table 1). However, in our evaluation study, it demonstrated substantially lower overhead. Despite this speed benefit, Uniform Sampling, Dead Reckoning, and Opening Window have fundamental limitations in controlling the growth of errors during compression (Section 4.3).

Both SQUISH-E(λ) and SQUISH-E(μ) demonstrate significantly higher compression speed than other algorithms that use SED. In particular, they are approximately 4–6 times faster than TD-TR and often faster than Douglas-Peucker which uses a less expensive error metric (spatial error). This benefit in compression speed is due to the use of a priority queue which enables both fast and effective removal of points (Section 3.1). Furthermore, whenever a point is removed from the priority queue, SQUISH-E needs to update information for at most two points. On the other hand, both Douglas-Peucker and TD-TR tend to recalculate, for each addition of a point, error values with respect to a large number of points (Section 2.2.2). SQUISH-E(μ) is slightly slower than SQUISH-E(λ) since it can further remove redundant points as long as this removal increases SED error below a tolerable bound.

Although there were significant differences in run-time performance between algorithms, we did not observe the same amongst different datasets (i.e., travel modes). In general, particularly in the case of Uniform Sampling and SQUISH-E, compression time is affected directly by the trajectory size and compression ratio rather than variations in speed and heading. In contrast, these characteristics of trajectories may have significant impact on the errors introduced during compression (Table 5 and Fig. 9).

4.3 Evaluation based on error metrics

This section experimentally compares trajectory compression algorithms across multiple metrics including SED, spatial, and speed errors. Figures 7 and 8 contrast

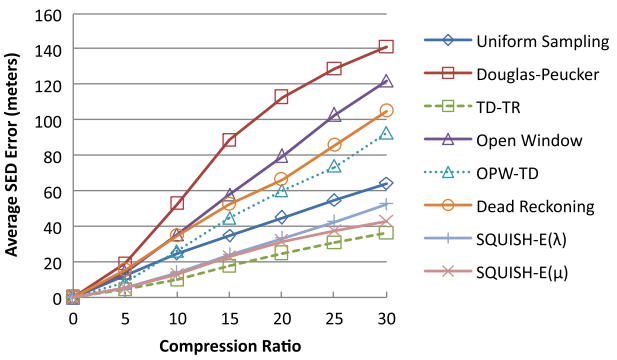


Fig. 7 Average SED error

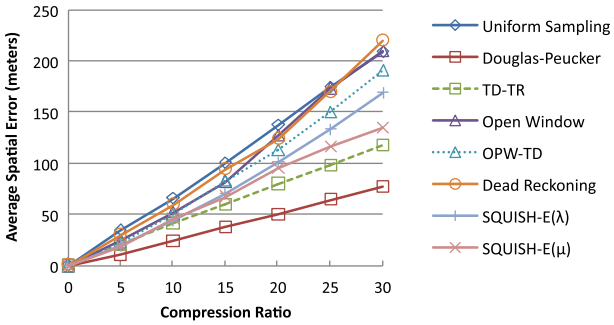


Fig. 8 Average spatial error

compression algorithms in terms of SED and spatial errors. These results are a composition from the three datasets mentioned in Section 4.1. In terms of overall accuracy, TD-TR and SQUISH-E(μ) outperform other algorithms with the differences between them being insignificant in most cases. TD-TR is slightly more accurate than SQUISH-E(μ) since it strives to minimize the maximum SED error by taking many points into account. As mentioned in Section 4.2, SQUISH-E is 4–6 times faster than TD-TR because it updates information for at most two points for each removal of a point. Unlike other algorithms, SQUISH-E also has the ability to compress trajectories while balancing both the compression ratio and error introduced during compression.

In Figs. 7 and 8, algorithms that are faster than SQUISH-E (Fig. 6) have substantial limitations in controlling errors during compression. For example, Uniform Sampling had about two times higher SED and spatial errors than SQUISH-E(μ). Surprisingly, both Dead Reckoning and Opening Window introduced larger SED and spatial errors than Uniform Sampling. These figures also illustrate that SED error has benefits over spatial error. For example, algorithms such as Douglas-Peucker and Opening Window, that are optimized for spatial error tend to introduce the largest SED errors (Fig. 7). On the other hand, algorithms that take into account SED error, including TD-TR, SQUISH-E(λ), SQUISH-E(μ), and OPW-TD keep spatial error at a relatively low level. As explained in Section 2.1.1, SED error has the benefit of incorporating temporal data into error calculation.

The compression algorithms examined in this study attempt to minimize specific error metrics (Table 1). Douglas-Peucker is optimized for spatial error, while TD-TR, a modification of Douglas-Peucker, is optimized for SED error. Douglas-Peucker and TD-TR enabled most accurate compressions in terms of spatial error and SED error, respectively. In the case of Opening Window, a significant improvement can be observed for OPW-TR compared to Opening Window in Fig. 7.

Figure 9 demonstrates significant differences between various trajectory compression algorithms, as well as the datasets explained in Section 4.1. For this result, all algorithms were compared at the common compression ratio of 10. As discussed in Section 4.1.4, the GeoLife dataset had the highest degree of error, followed by the Bus and NYMTC datasets. For this reason, most of the algorithms introduced the largest SED error when they are compressing trajectories from the GeoLife dataset. In contrast to other algorithms, TD-TR, SQUISH-E(μ), and SQUISH-E(λ) effectively kept SED error under control across these datasets.

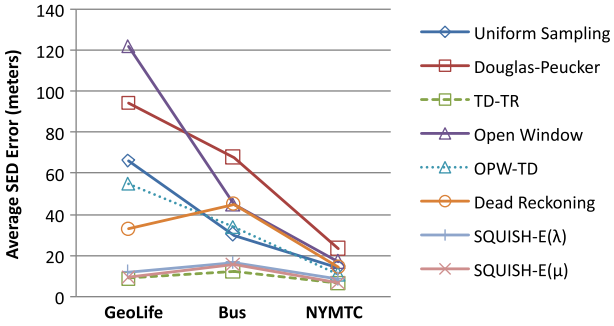


Fig. 9 Average SED error (per dataset)

Figure 10 provides an additional comparison which uses an ordered ranking of the algorithms in terms of average SED error. A ranking of 1 for a particular trajectory implies that the algorithm had the lowest SED error compared to the other algorithms. Similarly, an algorithm that had a ranking of 8 had the highest SED error among the 8 algorithms. The results presented in Fig. 10 were obtained over all of the three datasets mentioned in Section 4.1 and across various compression ratios that ranged from 5 to 30. TD-TR, SQUISH-E(μ) and SQUISH-E(λ) are shown to be the most accurate algorithms with a ranking of nearly 1, 2 and 3, respectively. Douglas-Peucker and Opening Window are the worst performers, presumably because they are not optimized for SED errors. Uniform Sampling had a better average ranking compared to Dead Reckoning.

Figure 11 shows the average speed error for each algorithm over various compression ratios. TD-TR and SQUISH-E(μ) were most accurate algorithms in terms of speed error. Uniform Sampling has the third best performance. The performance of the other algorithms was nearly identical. The effectiveness of SQUISH-E(μ) and TD-TR is due to the fact that they use temporal information in deciding which points to remove from or to add to the compressed representation.

Our previous SQUISH algorithm was vulnerable to error propagation at high compression ratios. Furthermore, it was not be able to provide provable guarantees on SED errors that the removal of points would introduce. SQUISH-E overcomes these problems using a different way of adjusting the priority of points. The correctness of this new approach is presented in Section 3.2 of this paper. Figure 12 shows the difference between these two versions in terms of average SED error. SQUISH-E

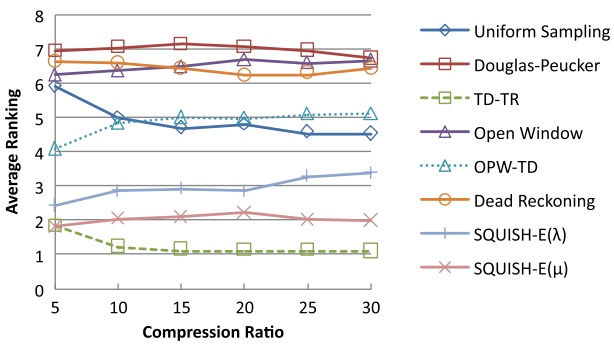


Fig. 10 Ranking in terms of average SED error

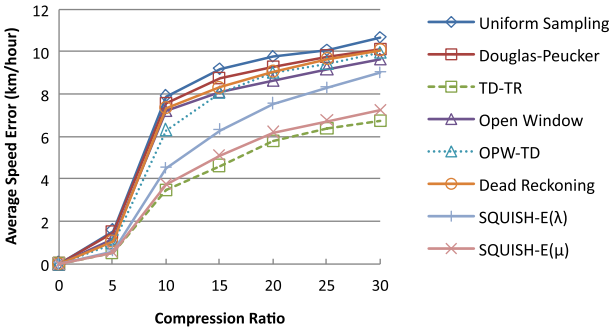


Fig. 11 Average speed error

enables a more tight control over the growth of SED error particularly under high compression ratios.

Figure 13 shows the distribution of SED errors for the compression algorithms at a compression ratio of 10. Each curve in the figure depicts the probability that the actual SED error will be be less than or equal to each error value. For example, the curve labeled "Douglas-Peucker" illustrates that the average SED error introduced by Douglas-Peucker was less than 100 m for about 80 % of the trajectories that were compressed. This distribution also indicates that there is no significant difference in terms of SED error between TD-TR and SQUISH-E(μ).

4.4 Discussion

As summarized in Table 1, trajectory compression algorithms have different characteristics in terms of compression parameters, compression speed and accuracy. Determining the most appropriate algorithm for a specific context requires understanding the tradeoffs that these algorithms offer in conjunction with the properties of the trajectory data to manage. Despite the results indicating significant differences in accuracy amongst the compression algorithms, the clear overall winners are TD-TR and SQUISH-E. Between these two algorithms, SQUISH-E compresses trajectories at a much higher speed than TD-TR.

Most applications require a balance between accuracy, performance and usability. For example, applications that have very short sampling intervals over long durations

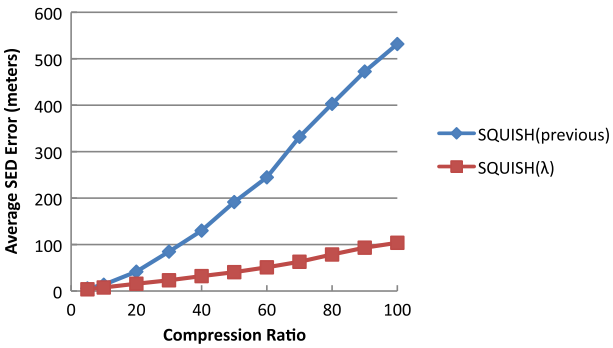


Fig. 12 Comparison of the previous version of SQUISH and new version SQUISH-E(λ)

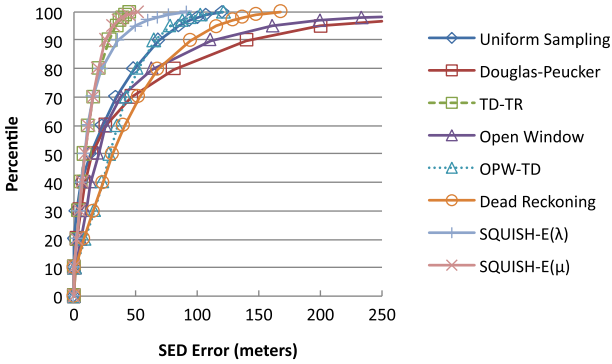


Fig. 13 Cumulative distribution of SED errors

might be willing to substitute accuracy for faster processing or space efficiency. SQUISH-E allows users to control compression with respect to both compression ratio and accuracy. This flexibility which is not supported by other algorithms fulfills an important niche in the literature.

5 Conclusions and future work

This paper presents a new approach to trajectory compression. This approach, called SQUISH-E (Spatial QUalIty Simplification Heuristic - Extended), inserts points from a trajectory into a priority queue where the priority of each point is set to an upper bound on the error that the removal of that point would introduce. In this way, it can quickly remove points while effectively bounding the growth of error caused by the removal of points. This technique also allows users to control compression while striking a balance between compression ratio and accuracy.

Another contribution of this paper is a comprehensive evaluation study which measures the effectiveness of trajectory compression algorithms in terms of performance and various error metrics. In this study, SQUISH-E achieves most accurate compression within a substantially shorter time than other techniques. This study also analyzed characteristics of real-world trajectory data and their impact on the performance of trajectory compression algorithms.

Our future work includes determining the effectiveness of compression on common spatial applications such as modeling of traffic flow, identification of congestion bottlenecks, and detection of speeding violation hot-spots. One possible approach for improving the quality of trajectory compression would be to use knowledge of the road network. This research would require fast detection of deviations from the road network and efficient storage of information that captures the behaviour of moving objects. A key goal in this research would be to enable a drastically smaller compressed representation with low computation overhead and reduced error.

Acknowledgements This work is supported in part by the National Science Foundation under CAREER award IIS-1149372 and the Research and Innovative Technology Administration of the U.S. Department of Transportation through the Region 2 – University Transportation Research Centers Program.

References

1. Canalys (2007) Worldwide mobile navigation device market more than doubles. Technical report, Canalys Research Release
2. Canalys (2009) North America overtakes EMEA as largest satellite navigation market. Technical report, Canalys Research Release
3. Meratnia N, de By RA (2004) Spatiotemporal compression techniques for moving point objects. In: Proceedings of the 9th international conference on extending database technology (EDBT), pp 765–782
4. Abdelguerfi M, Givaudan J, Shaw K, Ladner R (2002) The 2-3TR-tree, a trajectory-oriented index structure for fully envolving valid-time spatio-temporal datasets. In: Proceedings of the 10th SIGSPATIAL international conference on advances in geographic information systems (ACM-GIS), pp 29–34
5. Agarwal PK, Guibas LJ, Edelsbrunner H, Erickson J, Isard M, Har-Peled S, Hershberger J, Jensen C, Kavraki L (2002) Algorithmic issues in modeling motion. ACM Comput Surv 34:550–572
6. Zhu H, Su J, Ibarra OH (2002) Trajectory queries and octagons in moving object databases. In: Proceedings of the 11th conference on information and knowledge management (CIKM), pp 413–421
7. Prior-Jones M (2008) Satellite communications systems buyer's guide. British Antarctic Survey
8. Giannotti F, Nanni M, Pinelli F, Pedreschi D (2007) Trajectory pattern mining. In: Proceedings of the 13th international conference on knowledge discovery and data mining (ACM-KDD), pp 330–339
9. Muckell J, Hwang J-H, Lawson CT, Ravi SS (2010) Algorithms for compressing GPS trajectory data: an empirical evaluation. In: Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems (ACM-GIS), pp 402–405
10. Muckell J, Hwang J-H, Patil V, Lawson CT, Ping F, Ravi SS (2011) SQUISH: an online approach for GPS trajectory compression. In: Proceedings of the 2nd international conference on computing for geospatial research and applications (COM.Geo), pp 13.1–13.8
11. Potamias M, Patroumpas K, Sellis T (2006) Sampling trajectory streams with spatio-temporal criteria. In: Proceedings of the 18th international conference on scientific and statistical database management (SSDBM), pp 275–284
12. Harper JG (1991) Traffic violation detection and deterrence: implications for automatic policing. Appl Ergon 22(3):189–197
13. Karpinski M, Senart A, Cahill V (2006) Sensor networks for smart roads. In: Proceedings of the 4th IEEE conference on pervasive computing and communications workshops (PerCom 2006 Workshops), pp 306–310
14. Douglas DH, Peucker TK (1973) Algorithms for the reduction of the number of points required to represent a line or its caricature. Can Cartogr 10(2):112–122
15. Hershberger J, Snoeyink J (1992) Speeding up the Douglas-Peucker line simplification algorithm. In: Proceedings of the 5th international symposium on spatial data handling (SDH), pp 134–143
16. Keogh EJ, Chu S, Hart D, Pazzani MJ (2001) An online algorithm for segmenting time series. In: Proceedings of the 2001 IEEE international conference on data mining (ICDM), pp 289–296
17. Trajcevski G, Cao H, Scheuermann P, Wolfson O, Vaccaro D (2006) On-line data reduction and the quality of history in moving objects databases. In: Proceedings of the 5th ACM international workshop on data engineering for wireless and mobile access (MobiDE), pp 19–26
18. Bellman RE (1961) On the approximation of curves by line segments using dynamic programming. Commun ACM (CACM) 4(6):284
19. Muckell J, Hwang JH, Lawson CT, Ravi SS (2010) Algorithms for compressing GPS trajectory data: an empirical evaluation. Technical Report SUNYA-CS-10-06, CS Department, University at Albany – SUNY
20. Feldman D, Sugaya A, Rus D (2012) An effective coreset compression algorithm for large scale sensor networks. In: Proceedings of the 11th international conference on information processing in sensor networks (IPSN), pp 257–268
21. Agarwal P, Har-Peled S, Varadarajan K (2005) Geometric approximation via coresets. Technical report, Computer Science Department, Duke University
22. Schmid F, Richter K-F, Laube P (2009) Semantic trajectory compression. In: Proceedings of the 11th international symposium on advances in spatial and temporal databases (SSTD), pp 411–416

23. Lin CY, Chen HC, Chen YY, Lee WC, Chen LJ (2010) Compressing trajectories using inter-frame coding. Technical Report TR-IIS-10-007, Institute of Information Science
24. Kaul S, Gruteser M, Rai V, Kenney J (2010) On predicting and compressing vehicular GPS traces. In: Proceedings of the 2010 IEEE international conference on communications Workshops (ICC Workshops), pp 1–5
25. Potamias M, Patroumpas K, Sellis T (2006) Amnesic online synopses for moving objects. In: Proceedings of conference on information and knowledge managment (CIKM), pp 784–785
26. Potamias M, Patroumpas K, Sellis T (2007) Online amnesic summarization of streaming locations. In: Proceedings of the 10th international symposium on advances in spatial and temporal databases (SSTD), pp 148–166
27. Zheng Y, Li Q, Chen Y, Xie X, Ma W-Y (2008) Understanding mobility based on GPS data. In: Proceedings of the 10th international conference on ubiquitous computing (UbiComp), pp 312–321
28. Zheng Y, Zhang L, Xie X, Ma W-Y (2009) Mining interesting locations and travel sequences from GPS trajectories. In: Proceedings of the 18th international conference on world wide web (WWW), pp 791–800
29. Lawson CT, Mallia ME (2010) Understanding commuter patterns and behavior: an analysis to recommend policies aimed at reducing vehcile use. Technical report, The New York State Energy and Research and Development Authority (NYSERDA)
30. Lawson CT, Chen C, Gong H, Karthikeyan S, Kornhauser A (2009) GPS pilot project: phase four. Technical report, New York Metropolitan Transportation Council
31. Robusto CC (1957) The Cosine-Haversine formula. Am Math Mon 64(1):38–40

Jonathan Muckell graduated with his Ph.D. in Informatics from the University at Albany in 2013. He received his MS in Computer & Systems Engineering from Rensselaer Polytechnic Institute in 2008, and his BS in Computer Science & Mathematics from St. Lawrence University in 2006. Jonathan's dissertation received the University at Albany Distinguished Dissertation Award for his research on reducing the storage and computational demands required to analyze the large volumes of data generated by Geographic Positioning Systems (GPS).

Paul W. Olsen Jr. received his BA degree from State University of New York at Albany in 2010 with a double major in Philosophy and Computer Science. His areas of interest include design and analysis of algorithms, social network analysis, and database systems. He is currently pursuing a PhD in Computer Science at the State University of New York at Albany.

Jeong-Hyon Hwang is an assistant professor in the Department of Computer Science at the University at Albany – State University of New York. He received his BS degree in Computer Science from Korea University in 1998 and PhD in Computer Science from Brown University in 2008. Jeong-Hyon's research interests include databases and distributed systems. Jeong-Hyon is a recipient of the NSF CAREER award and 2005 ACM SIGMOD best demonstration award.

Catherine T. Lawson is the Department Chair and an Associate Professor in the Department of Geography and Planning as well as an Affiliated Faculty member of the Department of Informatics, in the College of Computing and Information at the University at Albany – State University of New York. She received her BA in Accounting and Economics from Western Washington University, Bellingham, WA, in 1988. She received her MA in Economics in 1995, MURP in Urban Planning in 1997, and PhD in Urban Studies/Regional Science, in 1998, from Portland State University (PSU), Portland, Oregon. Her research interests include advanced uses of archived intelligent transportation systems (ITS) data and spatial analysis/geographic information systems (GIS) applications for transportation planning and analysis for freight, transit, and passenger travel.

S. S. Ravi received his BE and ME degrees from the Indian Institute of Science, Bangalore, India, in 1977 and 1979 respectively. He received his PhD degree in Computer Science from the University of Pittsburgh, Pittsburgh, PA, in August 1984. Since September 1984, he has been with the Computer Science Department at the University at Albany – State University of New York, where he is currently a Distinguished Teaching Professor. His areas of interest include design and analysis of algorithms, social network analysis, data mining, fault-tolerant computing, wireless networks and discrete dynamical systems.