Simplification of Trajectory Streams

Progress Report

Yeung Sin Chun

scyeung@connect.ust.hk

December 14, 2025

Abstract

The ubiquitous use of GPS sensors has enabled real-time tracking of vehicles, which in turn enables the collection of massive trajectory data. Yet, for a massive stream, sending all vertices may be highly wasteful since only a small percentage of points on the trajectory are significant to maintain the shape of the original stream. While there are previous algorithms that does trajectory simplifications, few of them offers error guarantee using Fréchet distance. The aim of this project is to implement and benchmark a new streaming algorithm with a theorectical guarantee on Fréchet distance. We benchmark the algorithm in terms of the number of points in the simplified curve and the Fréchet distance achieved.

Introduction

Trajectory stream simplification is a critical task in the era of ubiquitous GPS tracking. As vehicles, mobile devices, and sensors generate massive volumes of location data in real-time, transmitting every single data point becomes bandwidth-inefficient. Much of this data is redundant. For instance, a vehicle moving in a straight line generates many points that contribute little to the trajectory's overall shape. Simplification reduces this data volume while preserving the essential geometric features, enabling faster transmission, and more efficient real-time analytics.

While many software systems perform on-the-fly simplification, few algorithms offer rigorous quality guarantees that satisfy streaming requirements. In this project, we explore a streaming algorithm described in[1]. However, we will only study the algorithm in the context of $\mathbb{R}^2$ because of the complexity of implementing the algorithm in higher dimension.

For user-defined parameters $\varepsilon \in (0, 1)$ and error bound $\delta > 0$, the algorithm constructs a simplified curve σ in $\mathbb{R}^2$ that satisfies two key guarantees. First, the simplified curve is "close" to the original curve such that at any prefix of the original curve $\tau[v_1, v_i]$, the simplified curve σ satisfies $d_F(\sigma, \tau[v_1, v_i]) \leq (1 + \varepsilon)\delta$. Second, the size of the simplified curve satisfies $|\sigma| \leq 2 \cdot \text{opt} - 2$ at any point during the algorithm, where opt is the minimum number of vertices required to achieve a Fréchet error of at most δ for the current prefix of the trajectory. The algorithm uses working storage of $O(\varepsilon^{-4})$ and each vertex in the original curve is processed in $O(\varepsilon^{-4} \log \frac{1}{\varepsilon})$ time in $\mathbb{R}^2$.

Fréchet Distance

The Fréchet distance is a measure of similarity between two curves that takes into account the location and ordering of the points along the curves. Let S be a metric space. A curve A in S is a continuous map from the unit interval into S, i.e., $A : [0, 1] \to S$. A reparameterization α of $[0, 1]$ is a continuous, non-decreasing, surjection $\alpha : [0, 1] \to [0, 1]$.

Let A and B be two continuous curves in S. The Fréchet distance $d_{F(A,B)}$ is defined as the infimum over all reparameterizations α and β of $[0, 1]$ of the maximum distance between $A(\alpha(t))$ and $B(\beta(t))$ for $t \in [0, 1]$. Formally:

$$d_{F(A,B)} = \inf_{\alpha,\beta} \max_{t\in[0,1]} d(A(\alpha(t)), B(\beta(t)))$$

where d is the distance metric in S. We adopt the usual Euclidean distance.

Intuitively, this metric is often illustrated using the "dog-walking" analogy: imagine a person walking along curve A and a dog walking along curve B. Both can control their speed but cannot move backwards. The Fréchet distance corresponds to the minimum length of the leash required to connect the dog and the person throughout their entire walk.

We will use the implementation in [2] for measuring Fréchet distance.

Other works

Trajectory simplification has been extensively studied in both batch and streaming contexts. Batch algorithms, which process the complete trajectory history, typically achieve a better trade-off between compression ratio at the cost of having a higher storage requirement. In contrast, streaming algorithms simplify the data with limited working storage. This difference is crucial in practice since batch algorithms can only be performed on the server side so a larger bandwidth is needed for data transmission to the server, whereas streaming algorithms simplify the data on-the-fly so they're more bandwidth-efficient.

The paper in [3] provides a comprehensive overview of existing trajecotry simplification algorithms as well as their implementations. One notable batch-mode algorithm is **DP**, which we use to benchmark our algorithm against. Other algorithms haven't been compared against because of limited time during this semester and will be continued in next semester.

Implementation

We implement the algorithm in [1] in C++ with a QT viewer for visualization. The source code of the algorithm is available for public[1]. We use the dataset provided by the paper [4] and [5] for testing and benchmarking. This dataset comprises GPS trajectories from 10,357 taxis in Beijing, collected between February 2 and February 8, 2008. It contains approximately 15 million data points, covering a total distance of 9 million kilometers.

The core of our implementation resides in `simplify.cpp`.

High-level idea of the algorithm

Define the error region of a point v_a by B_{v_a}, which contains all points with a distance at most d with v_a. The algorithm attempts to find the longest sequence of vertices $v_1, ..., v_i$ such that a single line segment can stab the error regions B_{v_a} for all $a \in [1, i]$ in order. To manage storage complexity, we approximate B_{v_a} using a convex hull of a set of grid points G_{v_a}, denoted as $\mathrm{conv}\left(G_{v_a}\right)$. Here, G_{v_a} is the set of grid squares that have non-empty intersection with in B_{v_a}. We restrict the starting point of the segment to a set of grid points P within the initial error region $\mathrm{conv}\left(G_{v_1}\right)$.

For each candidate starting point $p \in P$, we maintain a structure $S_a[p]$ representing the set of valid endpoints for a segment starting at p that stabs all regions up to v_a. This structure is updated inductively:

$$S_{a+1}[p] = \mathrm{conv}\left(G_{v_{a+1}}\right) \cap F(S_a[p], p)$$

where $F(S_a[p], p)$ is the region illuminated by p through the "window" $S_a[p]$. Effectively, $S_a[p]$ contains all points x in the current error region such that the segment px is a valid simplification for the prefix $v_1, ..., v_a$.

When $S_{i+1}[p]$ becomes empty for all p, it implies no segment starting from P can extend to v_{i+1}. The algorithm then outputs a valid segment pq from the previous step (where $q \in S_{i[p]}$)

---

[1] https://github.com/yeungsinchun/Simplification-of-Trajectory-Streams

and restarts the process from v_{i+1}, resetting P to points within the new initial error region $\operatorname{conv}\left(G_{v_{i+1}}\right)$. The final simplified curve σ is the concatenation of these output segments.

$\operatorname{conv}\left(G_{v_i}\right)$ $\quad$ $F(S_a[p], p)$

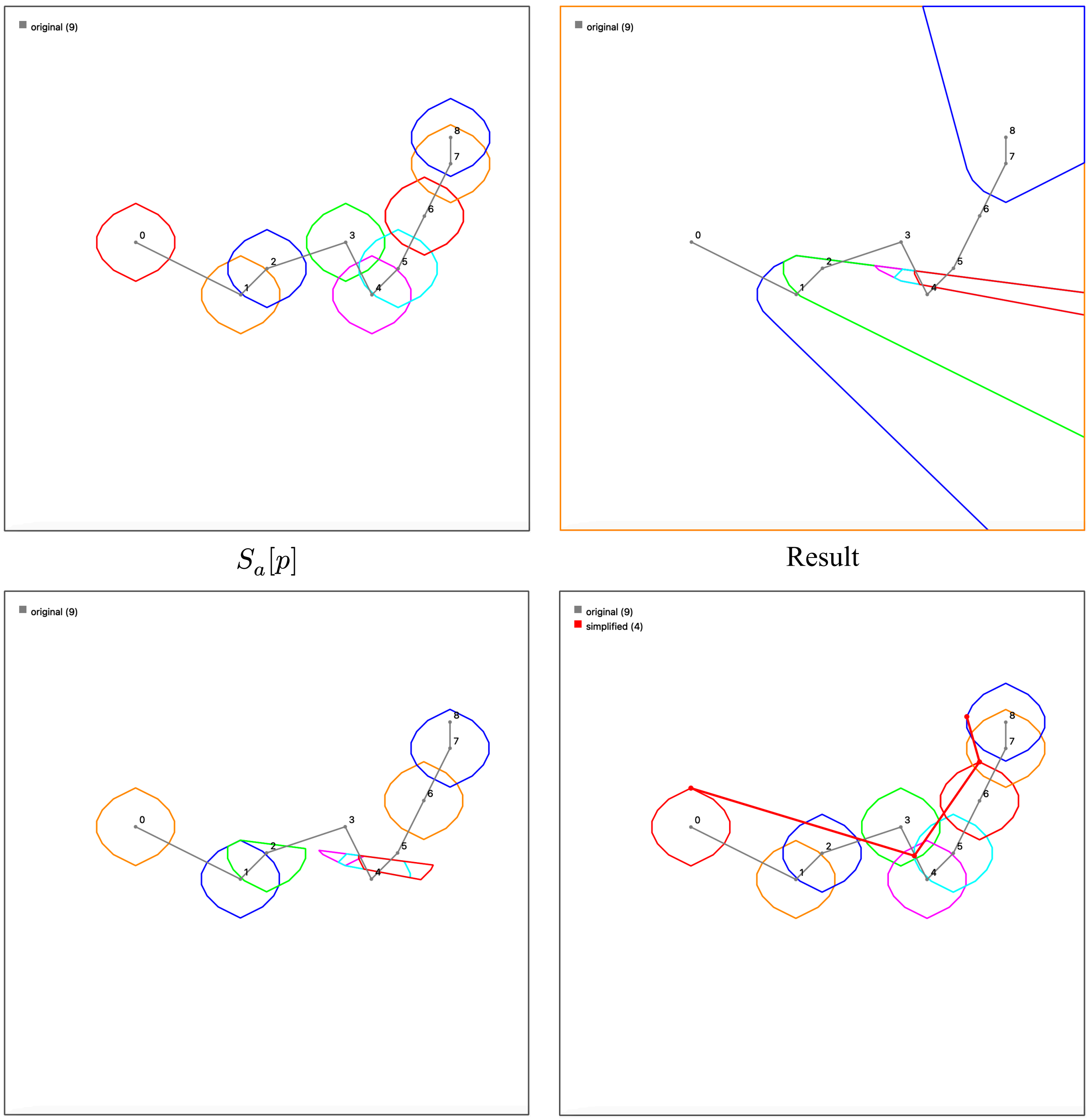

Figure 1: The figure shows $\operatorname{conv}\left(G_{v_i}\right)$, $F(S_a[p], p)$, and $S_a[p]$ where p is some point in $\operatorname{conv}\left(G_{v_1}\right)$. In this example, $S_6[p]$ becomes empty, so a simplified segment will be drawn from p to some point in $S_5[p]$. Then, the simplification continues the same way from v_6. Here, as shown in the result, the algorithm constructs one segment stabbing $\operatorname{conv}\left(G_{v_1}\right)$ up to $\operatorname{conv}\left(G_{v_5}\right)$ and another stabbing $\operatorname{conv}\left(G_{v_6}\right)$ up to $\left(G_{v_8}\right)$.

Evaluation

To evaluate the performance of our streaming simplification algorithm, we benchmark it against the Douglas-Peucker (DP) algorithm, a widely used batch simplification method. While DP is not a streaming algorithm, it serves as a strong baseline for compression quality. We use the Beijing Taxi Dataset, a typical trajectory and the associated simplified curve (both DP and our approach) are shown in the image below:

Figure 2: A typical trajectory

Methodology

Our benchmarking strategy is designed to compare the two algorithms under two distinct constraints: size reduction and error reduction.

For a given trajectory, we first run the benchmark algorithm (DP) to obtain a simplified curve of size S and measure its Fréchet distance D from the original input curve. We then configure our streaming algorithm with an error bound $\delta = \frac{D}{1+\varepsilon}$ for various values of ε in

$\{0.25, 0.5, 0.75\}$ and pick the simplified trajectory with the least points. This construction ensures that our algorithm will obtain a Fréchet distance not exceeding D.

1. **Size Reduction**: We measure the size S' of the output curve produced by our algorithm. If $S' < S$, our algorithm has achieved a more compact representation.
2. **Error Reduction**: In cases where $S' < S$, we can further relax the compression to improve accuracy. We decrease the error bound δ (which increases the output size) until the new output size is approximately equal to the benchmark size S. We then measure the new Fréchet distance D' between the input and our output curve. If $D' < D$, our algorithm provides a more accurate representation for the same storage cost.

Results

We conducted this evaluation on a subset of the Beijing taxi dataset using around 100 trajectories because both the calculation of Fréchet distance and our algorithm takes substantial time. The results demonstrate the effectiveness of the streaming algorithm:

In 56.1% of the test cases, our algorithm produced a smaller curve size ($S' < S$) while maintaining the stricter error bound. On average, when a reduction was achieved, the number of points was reduced by 30.27%.

Bibliography

[1] S.-W. Cheng, H. Huang, and L. Jiang, “Simplification of Trajectory Streams,” 2025, [Online]. Available: https://arxiv.org/abs/2503.23025

[2] S. Har-Peled, B. Raichel, and E. Robson, “The Fréchet Distance Unleashed: Approximating a Dog with a Frog,” p. , 2024, doi: 10.48550/arXiv.2407.03101.

[3] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. T. Shen, “Trajectory simplification: an experimental study and quality analysis,” *Proc. VLDB Endow.*, vol. 11, no. 9, pp. 934–946, May 2018, doi: 10.14778/3213880.3213885.

[4] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," 2011.

[5] J. Yuan *et al.*, "T-drive: driving directions based on taxi trajectories," pp. 99–108, 2010.