TrajMetrix: A Trajectory Compression Benchmarking Framework[*]

Kyuseo Park[†], Jeremy Birnbaum[†], Paul Olsen Jr.[†], Yuchao Ma[†], Jayadevan Vijayan[†], S. S. Ravi[†], Jeong-Hyon Hwang[†], Jonathan Muckell[‡], Catherine T. Lawson[§]
University at Albany – State University of New York, USA
{kpark, jbirn, polsen, yma, appu, ravi, jhh}@cs.albany.edu,
jonmuckell@gmail.com, lawsonc@albany.edu

ABSTRACT

Trajectory compression algorithms enable efficient transmission, storage, and processing of trajectory data by eliminating redundant information. While a large number of compression algorithms have been developed, there is no comprehensive and convenient benchmarking system for evaluating these algorithms. We will demonstrate TrajMetrix, our system that meets the above need. We will show how TrajMetrix can be used to gain insights into the benefits and drawbacks of various compression algorithms given different compression requirements.

From the knowledge attained by using TrajMetrix, we developed SQUISH-E (Spatial QUalIty Simplification Heuristic - Extended). This algorithm uses a priority queue to preferentially remove points based on the error introduced by their removal. Through live demonstrations that use both synthetic and real data sets, we will show the ability of SQUISH-E to effectively bound compression error with low computational overhead.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Experimentation, Measurement, Performance

[*]This work has been supported by the National Science Foundation under CAREER award IIS-1149372 as well as the Research and Innovative Technology Administration of the U.S. Department of Transportation through the Region 2 - University Transportation Research Centers Program.

[†]Department of Computer Science

[‡]Department of Informatics

[§]Department of Geography and Planning

SIGSPATIAL '13, Nov 05-08 2013, Orlando, FL, USA
ACM 978-1-4503-2521-9/13/11.
http://dx.doi.org/10.1145/2525314.2525326

Keywords

trajectory, compression, benchmarking

1. INTRODUCTION

The proliferation of GPS-equipped mobile devices has led to the rapid production of trajectory data. To efficiently store and process these data, a variety of trajectory compression algorithms have been developed [1, 2, 3, 4, 5, 9, 10]. These algorithms have different trade-offs in terms of compression time and compression ratio, thereby making it difficult to select the most suitable algorithm in practical situations. We are not aware of any standardized benchmarking framework for comparing trajectory compression algorithms.

We developed TrajMetrix, a system for efficiently and accurately evaluating trajectory compression algorithms. TrajMetrix allows users to easily examine various compression algorithms, evaluation metrics, and trajectory data sets. It also facilitates the incorporation of new compression algorithms and metrics. TrajMetrix evaluates compression algorithms on a fair basis despite their inherent differences (e.g., one algorithm may maximize compression ratio given an error tolerance, whereas another may achieve a target compression ratio while minimizing one type of error). TrajMetrix also provides trajectory generators that take advantage of realistic traffic flow models or that enable a tight control over the variation of traffic speed and direction. Furthermore, TrajMetrix supports large-scale evaluations that fully utilize the collective capability of a server cluster.

The lessons learned by using TrajMetrix enabled us to develop SQUISH-E (Spatial QUalIty Simplification Heuristic - Extended) [7], an enhanced version of our previous SQUISH algorithm [6]. SQUISH-E uses a priority queue of points, where the priority of each point is set to an *upper bound* of the error that would be introduced by removing that point. Points of the lowest priority are then removed from the queue as long as a user-specified error requirement is met. In this way, SQUISH-E enables fast and effective compression with provable guarantees on *compression error*.

Our demonstrations will exhibit the aforementioned benefits of TrajMetrix and SQUISH-E. Using both synthetic and real data sets, we will also show the different benefits of trajectory compression algorithms and their dependence on the characteristics of trajectory data.

The remainder of this demonstration proposal is organized as follows: Section 2 presents an overview of TrajMetrix. Section 3 explains the SQUISH-E algorithm. Section 4 describes the demonstration environment, interface, and specific demonstration scenarios.

2. SUMMARY OF TRAJMETRIX

This section summarizes the evaluation metrics (Section 2.1), compression algorithms (Section 2.2), and data generators (Section 2.3) supported by TrajMetrix as well as the architecture of TrajMetrix (Section 2.4).

2.1 Metrics

A trajectory of length n contains 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. TrajMetrix provides metrics for evaluating compression algorithms that produce, given a trajectory T, a compressed representation T' of T. These metrics include *compression time* (i.e., the amount of time it takes to compress trajectory data), *compression ratio* (i.e., the size of the original trajectory T divided by the size of the compressed representation T'), and *error metrics* defined as follows: (1) The *spatial error* of T' with respect to a point P_i in T is defined as the distance between (x_i, y_i) and (x'_i, y'_i), where x'_i and y'_i are obtained from T' as estimates of x_i and y_i. (2) While spatial error ignores temporal data, the *Synchronized Euclidean Distance (SED)* between point $P_i(x_i, y_i, t_i)$ and its estimation $P'_i(x'_i, y'_i, t_i)$ is defined as the distance between (x_i, y_i) and (x'_i, y'_i), where x'_i and y'_i are obtained from T' for time point t_i. (3) 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.

2.2 Compression Algorithms

TrajMetrix supports various trajectory compression algorithms [1, 3, 4, 5, 9, 10] (refer to [7] for details of these algorithms). Uniform Sampling achieves a target *compression ratio* whereas Douglas-Peucker and Opening Window ensure *spatial error* within a specified bound. Dead Reckoning, TD-TR, and OPW-TR limit *SED* under a specified bound. Our SQUISH-E algorithm is summarized in Section 3. New compression techniques can also be incorporated into TrajMetrix if they are implemented using our API [8].

2.3 Trajectory Generators

In addition to supporting real-world data sets, TrajMetrix provides two trajectory generators. The first one, called Gaussian Generator, varies the speed and direction of moving objects according to statistics obtained from actual trajectories. This generator enables a tight control over variations in speed and direction. Therefore, it can be effectively used for evaluating the reliability of trajectory compression algorithms. Another trajectory generator overcomes a limitation of the Brinkhoff generator which cannot produce long, densely sampled trajectories (refer to [8] for further details of these two generators).

2.4 Architecture of TrajMetrix

TrajMetrix consists of a master server which controls the overall system and worker servers that evaluate compression algorithms for a subset of trajectories (Figure 1). As further explained in Section 4.3, the master reads a configuration file that specifies the location of trajectory data, the compression algorithms to evaluate (Section 2.2), as well as evaluation metrics (e.g., compression time, SED) and result aggregation methods (e.g., average, maximum). After parsing the configuration file, the master reads the relevant trajectory data from the file system and then assigns evaluation tasks to available workers in the cluster. Each evaluation task involves a subset of trajectories, a compression algorithm, evaluation metrics, aggregation methods, the target compression ratio (e.g., 5.0), and either a `DefaultExecutor` or a `GuaranteedCompressionRatioExecutor`.

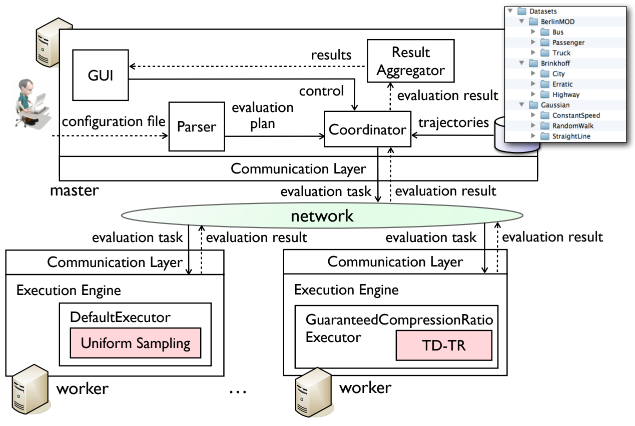


Figure 1: TrajMetrix Architecture

The `DefaultExecutor` and `GuaranteedCompressionRatioExecutor` are for accurately evaluating compression algorithms on a fair basis despite their differences. A `DefaultExecutor` runs an algorithm, such as Uniform Sampling, which compresses trajectories while guaranteeing a specified compression ratio. If the compression time is too short (e.g., 1 millisecond) to be considered a reliable measurement, the `DefaultExecutor` repeatedly runs the compression algorithm up to a predefined amount of time (e.g., 1 second) and then uses the average compression time obtained over these recent executions. For algorithms such as Douglas-Peucker and TD-TR, which ensure a specified error bound, the `GuaranteedCompressionRatioExecutor` varies the error bound using binary search until the compression ratio is within the desired range (e.g., between 4.5 and 5.5).

In addition to the compression time mentioned above, error metrics are measured by comparing each original trajectory and the compressed representation of that trajectory (Section 2.1). The error metric values are then aggregated for each result aggregation method (e.g., average, maximum). These aggregated results are then sent back to the master and then further aggregated before they are presented to the user. TrajMetrix has an extensible architecture that facilitates the incorporation of new compression algorithms, error metrics, and result aggregation methods (refer to [8] for further details).

3. SQUISH-E

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* of the SED error that would be introduced by removing that point [7]. SQUISH-E can remove a point with the lowest priority from Q in $O(\log |Q|)$ time, where $|Q|$ denotes the number of points stored in Q. Therefore, SQUISH-E usually achieves both fast and accurate compression.

Algorithm 1: SQUISH-E(T, λ, μ)

```
input   : trajectory $T$, lower bound $\lambda$ on compression
          ratio, and an additional parameter $\mu$
output  : trajectory $T'$
1  initialize $\beta$ to a small integer value; // Q's initial capacity
2  for each point $P_i \in T$ do
3      if $\frac{i}{\lambda} \geq \beta$ then
4          $\beta \leftarrow \beta + 1$; // increase the capacity of Q
5      set_priority($P_i, \infty, Q$); // enqueue $P_i$ with priority $\infty$
6      $P_i.\rho \leftarrow 0$;
7      if $i > 1$ then // $P_i$ is not the first point
8          $P_{i-1}.succ \leftarrow P_i$;
9          $P_i.pred \leftarrow P_{i-1}$;
10         adjust_priority($P_{i-1}, Q$); // Algorithm 3
11     if $|Q| = \beta$ then // Q is full
12         reduce($Q$); // Algorithm 2
13 while min_priority($Q$) $\leq \mu$ do // the lowest priority $\leq \mu$
14     reduce($Q$); // Algorithm 2
15 return trajectory $T'$ consisting of the points in $Q$
   sorted in their original order;
```

variable	description
Q	priority queue
β	capacity of Q
$P.pred$	P's closest predecessor among the points in Q
$P.succ$	P's closest successor among the points in Q
$P.\rho$	the maximum of the priorities that the neighboring points of P had when they were removed from Q

Table 1: Variables used in SQUISH-E

Algorithm 1 shows the operation of SQUISH-E. Table 1 summarizes the variables used in SQUISH-E. Given a trajectory T and parameters λ and μ, SQUISH-E compresses T while striving to minimize SED and achieve the compression ratio of λ (lines 1-12). Then, it further compresses T as long as this compression does not increase SED beyond μ (lines 13-14). Therefore, setting μ to 0 causes this algorithm to minimize SED, 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 (i.e., SQUISH-E maximizes compression ratio while keeping SED under μ).

In SQUISH-E, variable β stores the capacity of Q. Initially, β is set to a small integer value (line 1); we have chosen the value 4 in our implementation. 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 inserted into Q with a priority of ∞ (line 5). If P_i is not the first point (line 7), then P_i is registered as the closest successor of its previous point P_{i-1} (line 8), and P_{i-1} is registered as the closest predecessor of P_i (line 9). Then, the priority of P_{i-1} is adjusted to the SED error that would be introduced by its removal (line 10). When Q is full (i.e., $|Q| = \beta$), SQUISH-E reduces Q by removing a point with the lowest priority (lines 11 and 12). Reducing Q in this way ensures that Q keeps only $\beta - 1$ points and effectively limits the growth of SED error. Once all of the points in T are processed, SQUISH-E keeps removing a point with the lowest priority from Q until every point remaining in Q has a priority higher than μ (lines 13 and 14). SQUISH-E ensures that the resulting

Algorithm 2: reduce(Q)

```
input   : priority queue $Q$
1  $P \leftarrow$ remove_min($Q$); // lowest-priority point removed
2  $P.pred.succ \leftarrow P.succ$;
3  $P.succ.pred \leftarrow P.pred$;
4  $P.succ.\rho \leftarrow \max(priority(P), P.succ.\rho)$;
5  $P.pred.\rho \leftarrow \max(priority(P), P.pred.\rho)$;
6  adjust_priority($P.pred, Q$); // Algorithm 3
7  adjust_priority($P.succ, Q$); // Algorithm 3
```

Algorithm 3: adjust_priority(P, Q)

```
input   : point $P$, priority queue $Q$
1  if $P.pred \neq$ null and $P.succ \neq$ null then
2      $p \leftarrow P.\rho + \mathsf{SED}(P, P.pred, P.succ)$;
3      set_priority($P, p, Q$);
```

compression ratio is no less than λ and the SED error is no larger than the lowest priority of the points remaining in Q (refer to [7] for formal proofs).

Algorithms 2 and 3 show the details of reducing Q. This process first removes a point P with the lowest priority from Q. It then adjusts variables for P's predecessor and successor including their priorities (refer to [7] for further details).

4. DEMONSTRATION DETAILS

This section describes the demonstration environment (Section 4.1), interface (Section 4.2), and specific demonstration scenarios (Section 4.3).

4.1 System Setup

We will form a cluster of four Mac Minis, each of which has a 2.3GHz Quad-Core Intel i7 CPU, 8GB RAM, and a 1TB Serial ATA Drive. These machines will store trajectories obtained from both GPS devices and the data generators mentioned in Section 2.3.

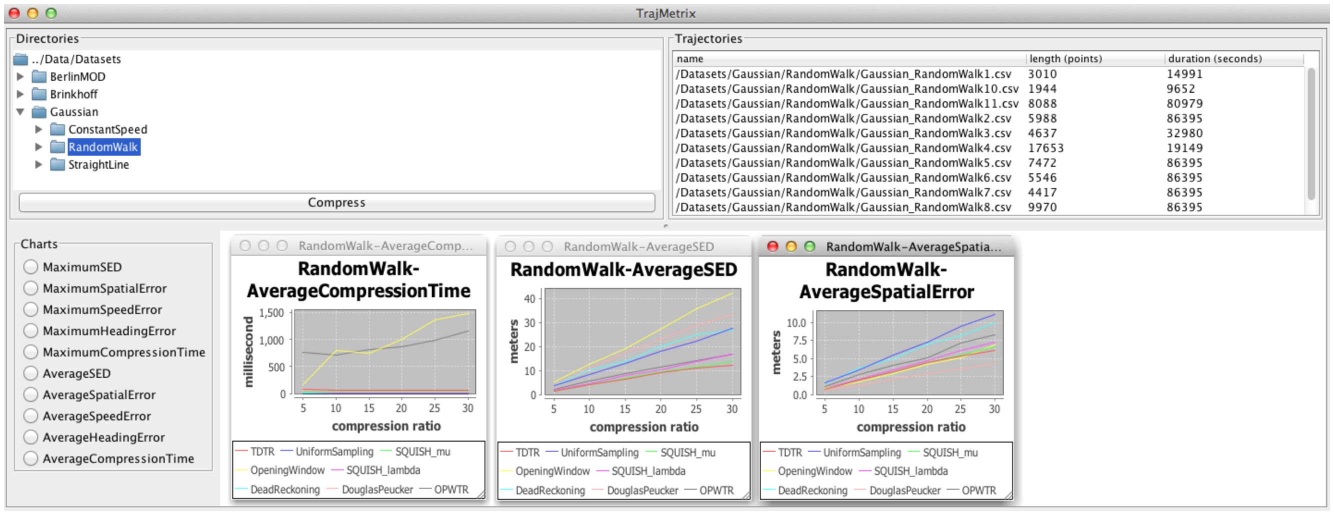
4.2 Demonstration Interface

Conference attendees will interact with TrajMetrix using the GUI shown in Figure 2. The *Directories* section of the GUI provides a directory hierarchy from which data sets can be selected. When a directory is chosen, details of the trajectories within that directory and its sub-directories will be shown in the *Trajectories* panel. If a user presses the *Compress* button, then the system will compress the selected trajectories and the *Charts* panel will show on the fly results as the evaluation proceeds.

4.3 Demonstration Scenarios

We will demonstrate the benefits of TrajMetrix and SQUISH-E as follows:

System Usability. We will demonstrate the steps of running the TrajMetrix master and workers and show that the overall benchmark can be easily set up using a configuration file. The sample configuration file in Figure 3 specifies the directory containing trajectory collections (line 1) and the directory for storing output trajectories (line 2). It also specifies the compression algorithms to evaluate for the compression ratios of 5.0, 10.0, 15.0, 20.0, 25.0, and 30.0 (lines 3-7). In this example, compression algorithms that achieve a



name	length (points)	duration (seconds)
/Datasets/Gaussian/RandomWalk/Gaussian_RandomWalk1.csv	3010	14991
/Datasets/Gaussian/RandomWalk/Gaussian_RandomWalk10.csv	1944	9652
/Datasets/Gaussian/RandomWalk/Gaussian_RandomWalk11.csv	8088	80979
/Datasets/Gaussian/RandomWalk/Gaussian_RandomWalk2.csv	5988	86395
/Datasets/Gaussian/RandomWalk/Gaussian_RandomWalk3.csv	4637	32980
/Datasets/Gaussian/RandomWalk/Gaussian_RandomWalk4.csv	17653	19149
/Datasets/Gaussian/RandomWalk/Gaussian_RandomWalk5.csv	7472	86395
/Datasets/Gaussian/RandomWalk/Gaussian_RandomWalk6.csv	5546	86395
/Datasets/Gaussian/RandomWalk/Gaussian_RandomWalk7.csv	4417	86395
/Datasets/Gaussian/RandomWalk/Gaussian_RandomWalk8.csv	9970	86395

Figure 2: TrajMetrix Graphical User Interface

```
1 input=~/TrajMetrix/Datasets/
2 output=~/TrajMetrix/Compressed/
3 executor=DefaultExecutor({UniformSampling,
4  SQUISH_lambda}, {5.0, 10.0, 15.0, 20.0, 25.0, 30.0})
5 executor=GuaranteedCompressionRatioExecutor({TDTR,
6  DouglasPeucker, OpeningWindow, OPWTR, DeadReckoning,
7  SQUISH_mu}, {5.0, 10.0, 15.0, 20.0, 25.0, 30.0}, 0.5)
8 metrics=({SED, SpatialError, SpeedError, HeadingError,
9  CompressionTime}, {Average, Maximum})
```

Figure 3: Example Configuration File

target compression ratio (i.e., SQUISH-E(λ), UniformSampling) will be evaluated using a `DefaultExecutor` (lines 3-4). Other algorithms that strive to maximize compression ratio under a certain error bound (e.g., Douglas-Peucker, SQUISH-E(μ), and TD-TR) will be evaluated using a `GuaranteedCompressionRatioExecutor` (lines 5-7). This example also specifies evaluation metrics and aggregation methods used for the benchmark (lines 8-9).

Result Reliability. We will demonstrate the ability of TrajMetrix to accurately measure the compression time of each algorithm through trials repeated for a predefined amount of time.

Evaluation Scalability. We will demonstrate that TrajMetrix can effectively reduce the overall evaluation time as it uses more machines.

Analysis of Compression Algorithms. We will demonstrate unique characteristics of trajectory compression algorithms. In particular, we will show that (1) both TD-TR and SQUISH-E achieve the smallest SED, while SQUISH-E is substantially faster than TD-TR, (2) Douglas-Peucker leads to the smallest spatial error, and (3) trajectories with significantly varying speed and direction cause large compression errors across all compression algorithms.

5. REFERENCES

[1] D. Douglas and T. Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Line or its Caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[2] R. Gotsman and Y. Kanza. Compact Representation of GPS Trajectories over Vectorial Road Networks. In *SSTD*, pages 241–258, 2013.

[3] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An Online Algorithm for Segmenting Time Series. In *ICDM*, pages 289–296, 2001.

[4] N. Meratnia and R. A. de By. Spatiotemporal Compression Techniques for Moving Point Objects. In *EDBT*, pages 765–782, 2004.

[5] J. Muckell, J.-H. Hwang, C. T. Lawson, and S. S. Ravi. Algorithms for Compressing GPS Trajectory Data: An Empirical Evaluation. In *SIGSPATIAL GIS*, pages 402–405, 2010.

[6] J. Muckell, J.-H. 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.

[7] J. Muckell, P. W. O. Jr., J.-H. Hwang, C. T. Lawson, and S. S. Ravi. Compression of Trajectory Data: A Comprehensive Evaluation and New Approach. Accepted to *GeoInformatica* (July 2013).

[8] J. Muckell, P. W. O. Jr, J.-H. Hwang, S. S. Ravi, and C. T. Lawson. A Framework for Efficient and Convenient Evaluation of Trajectory Compression Algorithms. Accepted to *COM.Geo*, 2013.

[9] M. Potamias, K. Patroumpas, and T. Sellis. Sampling Trajectory Streams with Spatio-temporal Criteria. In *SSDBM*, pages 275–284, 2006.

[10] 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.