SQUISH: An Online Approach for GPS Trajectory Compression

Jonathan Muckell
Dept. of Informatics
University at Albany–SUNY
Albany, NY 12222
jonmuckell@gmail.com

Vikram Patil
Dept. of Computer Science
University at Albany–SUNY
Albany, NY 12222
vp322396@albany.edu

Fan Ping
Dept. of Computer Science
University at Albany–SUNY
Albany, NY 12222
apping@cs.albany.edu

Jeong-Hyon Hwang
Dept. of Computer Science
University at Albany–SUNY
Albany, NY 12222
jhh@cs.albany.edu

Catherine T. Lawson
Dept. of Geography & Planning
University at Albany–SUNY
Albany, NY 12222
lawsonc@albany.edu

S. S. Ravi
Dept. of Computer Science
University at Albany–SUNY
Albany, NY 12222
ravi@cs.albany.edu

ABSTRACT

GPS-equipped mobile devices such as smart phones and in-car navigation units are collecting enormous amounts spatial and temporal information that traces a moving object's path. The popularity of these devices has led to an exponential increase in the amount of GPS trajectory data generated. The size of this data makes it difficult to transmit it over a mobile network and to analyze it to extract useful patterns. Numerous compression algorithms have been proposed to reduce the size of trajectory data sets; however these methods often lose important information essential to location-based applications such as object's position, time and speed. This paper describes the Spatial QUalIty Simplification Heuristic (SQUISH) method that demonstrates improved performance when compressing up to roughly 10% of the original data size, and preserves speed information at a much higher accuracy under aggressive compression. Performance is evaluated by comparison with three competing trajectory compression algorithms: Uniform Sampling, Douglas-Peucker and Dead Reckoning.

Categories and Subject Descriptors

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

General Terms

Algorithms and Applications

Keywords

GPS, Trajectories, Compression

1. INTRODUCTION

A **trajectory** is defined as a series of 3-tuple records consisting of the position (Latitude, Longitude), along with the temporal information (when the moving object was at the location). The stream of points defines a path or trajectory of a moving object over a period of time. The enormous prevalence and popularity of GPS devices has led to a growth market called Location-based services. These applications, often developed for mobile devices, utilize location information, such as the location of the person accessing the application, and commonly the location patterns of other users. Location-based service is an active area of research, encompassing applications such as supply-chain management, location detection and traffic modeling [11, 6, 8, 15, 9].

Three major problems currently exist in location-based applications that use trajectory data: storing, transmitting and processing the data. Storing the data is difficult because the shear volumes of data can quickly overwhelm available data storage. For instance, if data is collected at 10 second intervals, a calculation due to Meratnia and de By [10] shows that without any data compression, 1 Gb of storage capacity is required to store just over $4,000$ objects for a single day.

The cost of transmitting GPS data from mobile devices is the second main problem highlighting the need to compress GPS trajectory data. Sending a large amount of data can often be expensive and problematic. To illustrate this difficultly imagine a large fleet of vehicles equipped with both a GPS device and a transmission mechanism for sending the information. Typical transmission mechanisms include cellular transmission and/or satellite communication. Satellite transmission is more expensive, but has the advantage of not having coverage gaps which are inherent in cellular networks. The cost of sending large volume of data over remote networks can be prohibitively expensive, typically ranging from $5 to $7 per megabyte [14]. Therefore, 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. Compression algorithms that reduce the size of the data can significantly reduce this incremental cost.

Applications that detect patterns from trajectory data are

increasingly popular. However, the computational complexity of these algorithms makes it difficult to generate useful patterns when the data size is very large. Reducing the size of the input data has the potential to improve the running time without significantly hindering performance [5]. In essence, this involves compressing the data as a preprocessing step prior to running computationally expensive algorithms. Many applications require online processing and transmission of GPS trajectory data.

Finding solutions to the problems of storing, transmitting and processing GPS trajectories utilizing compression methods is an active area of research [1, 2, 20, 10, 12]. The most trivial compression method is Uniform Samplng which simply takes every ith point in the trajectory. Uniform sampling can reduce the storage requirements, but often results in significant information loss. Another common algorithm for compressing trajectories is the Douglas-Peucker algorithm [3] which is often used in cartographic and computer graphics applications. However, this method runs in batch mode, and therefore cannot be used in mobile applications that require processing GPS points as a data stream. Another algorithm compared in this study is called Dead Reckoning [16]. Unlike Douglas-Peucker, Dead Reckoning is an online method that runs in linear time. However, often times, applications require more precise approximations of the original trajectory than is achieved using Dead Reckoning.

In this study, we introduce a new method that prioritizes the most important points in a trajectory stream. This Spatial QUalIty Simplification Heuristic or SQUISH, is based of the priority queue data structure [17, 4]. Since SQUISH is an online method, it uses local optimization to select the best subset of points and permanently removes redundant or insignificant points from the original GPS trajectory. Drastic improvement in performance is observed in small to medium-sized compression ratios (e.g. containing 10% or more of the original points). We carry out a performance comparison between SQUISH and three other methods (Uniform Sampling, Douglas-Peucker and Dead Reckoning) using 1300 GPS trajectories derived from the Microsoft GeoLife data set [18, 19].

To summarize, our main contributions include:

- A new online compression algorithm SQUISH that offers improved performance across multiple error metrics at small to medium compression ratios.
- Performance evaluation is compared to popular algorithms for trajectory compression: Uniform Sampling, Douglas-Peucker and Dead Reckoning.
- Suggestions regarding how future algorithms could be developed to improve performance.

The next section describes previous work for compressing trajectories, along with error metrics for evaluating the amount of information lost in compressing the data. In section 3, the SQUISH method is described in detail. The evaluation of SQUISH and other algorithms is described in section 4. Recommendations for future work and conclusions are provided in section 5.

Algorithm	**Running Time**	(Batch or Online)
Uniform Sampling	$O(n)$	Online
Dead Reckoning	$O(n)$	Online
Douglas-Peucker	$O(n \log n)$	Batch
SQUISH	$O(n \log(\beta))$	Online

Table 1: Summary of GPS Trajectory Algorithms (β is the buffer size used by SQUISH)

2. PREVIOUS WORK

In this paper, four algorithms are compared, three of which have been previously introduced in the literature. This section discusses the three previously developed algorithms (Uniform Sampling, Douglas-Peucker and Dead-Reckoning), as well as error metrics that have been defined specifically to determine the effectiveness of spatio-temporal trajectory data.

Compression strategies can be classified into two forms, namely lossless and lossy compression. Lossless compression enables an exact reconstruction of the original data; that is, no information is lost due to compression. 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 the storage requirements while maintaining an acceptable degree of error. All four algorithm described in this work are lossy compression algorithms. Furthermore, all four algorithms use line segments consisting of a subset of the points that exist in the original trajectory.

Various algorithms exist in the literature to balance the tradeoff between accuracy and storage size. These algorithms can be logically grouped based on criteria such as batch vs. online processing and the computational complexity. Table 1 depicts whether the algorithms are batch or online, as well as the computational complexity. The algorithms used in this study were selected as a representative sample of GPS trajectory compression algorithms. Uniform sampling is a simple and fast method; Douglas-Peucker is a popular batch algorithm used for line simplification; while Dead Reckoning is an online algorithm that incorporates spatio-temporal information into the compression strategy.

2.1 Uniform Sampling Algorithm

The most native method of lossy compression, namely uniform sampling, offers the advantages of time efficiency and reduced storage requirements for geo-spatial data. This approach down-samples a stream of GPS data at fixed time intervals; that is, from the original data consisting of a series of time-stamped points (x, y, t), every ith point is kept in the compressed version, for some suitable integer i. The primary advantage of Uniform Sampling is that it runs in linear time, and is trivial to implement. The major disadvantage is drastic changes in speed and direction are not stored if the change occurs between sampled points.

2.2 Dead Reckoning Algorithm

Dead Reckoning is based on an approach that has been used in navigation for hundreds of years. The basic goal is to estimate a future position p' at time t' using the object's current position and velocity. The Dead Reckoning Algorithm used in this study was based on the work developed by Trajcevski and Cao [16]. This implementation of Dead Reckoning takes a single tolerance θ, that specifies the

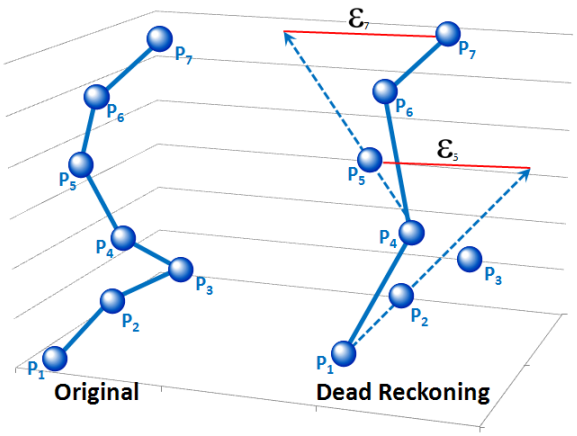

Figure 1: The Dead Reckoning algorithm estimates the future location of an object based on the current position and velocity.

maximum distance a future position can deviate from the estimated position. Let ϵ_i denote the distance of any point P_i on the original trajectory from its predicted location. If $\epsilon_i > \theta$, than P_i in the GPS stream is included in the compressed representation.

Figure 1 illustrates the processing of the Dead Reckoning algorithm. On the left, the original trajectory is shown, while on the right the compressed trajectory is shown using solid lines. In this algorithm, the GPS ray is projected out based on the current position and velocity of the GPS point. For instance the current location and velocity at P_1 is used to predict the future path of the trajectory, shown as the dashed arrow. Each subsequent point, P_2, P_3, etc... is compared with projected ray until the distance between the future point and the projected ray is beyond the user defined error threshold θ. Point P_5 is the first point beyond this error threshold, since $\epsilon > \theta$. When a point is beyond the user defined error tolerance, the point just before that location is added into the compressed representation. In the figure, point P_4 was included, since point P_5 violated the error condition. This process continues until each for all subsequent points. For instance, point P_7 distance is also beyond the error threshold and therefore P_6 is also included in the compression.

The computational complexity is $O(n)$, where n is the number of points in the trajectory. This complexity is due to the fact that the algorithm linearly processes each point; consecutively comparing each point to the predicted location. A major advantage of this algorithm is that it runs in online mode, unlike the Douglas-Peucker discussed next. The primary disadvantage is that Dead Reckoning is unable to set a specific error rate. Although Dead Reckoning typically performs better compared to Uniform Sampling and Douglas-Peucker, it is still susceptible to large worst-case sed and speed errors (As shown in section 4).

2.3 Douglas-Peucker Algorithm

The Douglas-Peucker Algorithm [3] is a popular heuristic, commonly used to fit a series of line segments to a curve, thereby reducing storage requirements. Often implemented in computer graphics applications, the Douglas-Peucker algorithm is applicable in a variety of geospatial applications. A common application is in reducing the number of points required to store state and county boundaries.

Douglas-Peucker is a line generalization algorithm, that recursively selects points from the original set of GPS trajectory points. A series of line segments is fitted to the original curve based on new points that are selected. The execution of the algorithm proceeds along the following steps. Initially, the first and last points in the trajectory are stored in the compressed version. Figure 2 shows Douglas-Peucker over four different time steps. At time t_0, the original trajectory is shown consisting of six points $P_1 - P_6$. The dotted lines indicate the original trajectory. The first step is to connect the end points A and P_6 together using a single line segment, shown at time t_1 as the solid line. The point with the greatest distance between the original trajectory and the line segment is included in the compression. For instance, point P_3 has the greatest distance at time t_1. Therefore, point P_3 is added to the compression. Three points or two line segments now exist in the compressed representation.

Douglas-Peucker is a recursive algorithm, repeating the process of adding the point that lies the maximum distance from the original line for each line segment. The process stops once the maximum distance for each line segment is less then some user defined error tolerance (i.e. $\epsilon < Tolerance$). Therefore, at time t_2 point P_2 is not added since the maximum distance is not greater than the user provided tolerance parameter. However, point P_5 is added since the distance is greater than the tolerance. Time step t_3 illustrates the final compressed approximation of the trajectory.

If the above algorithm is implemented in a straightforward manner, its worst-case running time is $O(n^2)$, where n is the number of original points. The running time can be improved to $O(n \log n)$ using a more complex approach involving convex hulls [7]. The primary advantage of Douglas-Peucker that the maximum error rate is bounded by some user-specified maximum error. The main drawback regarding Douglas-Peucker is that it is a batch algorithm; that is, the entire trajectory must be stored before compression. Therefore it is typically not suitable for mobile and real-time applications. Additionally, the user has no ability to set the desired compression rate.

2.4 Error Metrics

Algorithms for compressing GPS trajectories attempt to minimize one or more of the following error metrics: spatial distance, synchronized Euclidean distance (sed) and speed. GPS trajectories do not consist only of spatial data (latitude, longitude); crucially, the temporal component t, is also stored along with the spatial location (x, y). The three components define a series of time-stamped positions (x, y, t). Therefore, when evaluating the effectiveness of each algorithm, both spatial and temporal accuracy must be measured. Applications using stored GPS data, often require the preservation of the spatial component (where was the asset?), the temporal component (when was the asset at that location?), as well as velocity.

One way of measuring the difference between a GPS trace and its compressed version is to measure the perpendicular distance (Figure 3). The Douglas-Peucker algorithm attempts to minimize the maximum spatial distance error. Since the perpendicular distance does not incorporate temporal content, a more effective approach involves measuring

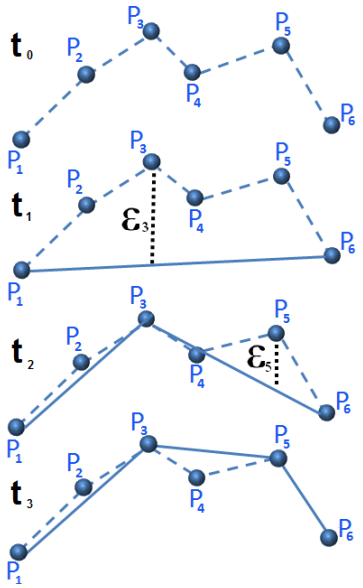

Figure 2: Douglas-Peucker Line Generalization Algorithm recursively approximate a curve by selecting the point furthest away from each line segment until the distance is below a user-specified tolerance.

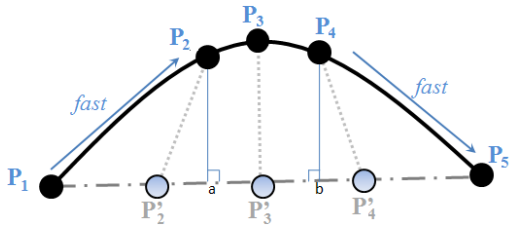

Figure 3: Synchronized Euclidean Distance measures the distance between the original and compressed trace at the same time. In contrast, line generalization algorithms ignore the temporal component and use simple perpendicular distance.

the distance at synchronized points. For instance, Figure 3 shows the distances at synchronized time points t_2, t_3 and t_4 (at time t_3, the time synchronized distance and the perpendicular distance coincide).

Synchronized Euclidean distance (sed) measures the distance between two points at identical time stamps [13]. In Figure 3, five time steps (t_1 through t_5) are shown. The simplified line (which can be thought of as the compressed representation of the trace) consists of only two points (P_{t_1} and P_{t_5}); thereby, it does not include points P_{t_2}, P_{t_3} and P_{t_4}. To quantify the error introduced by these missing points, distance is measured at identical time steps. Since three points were removed between P_{t_1} and P_{t_5}, the line is divided into four equal sized line segments using the three points P'_{t_2}, P'_{t_3} and P'_{t_4} for the purposes of measuring the error. The total error is measured as the sum of the distance between all points at the synchronized time instants, as shown below. (In the following expression, n represents the total number of points considered.)

$$sed = \sum_{i=1}^{n} \sqrt{(x_{ti} - x'_{ti})^2 + (y_{ti} - y'_{ti})^2}$$

2.5 Comparison of GPS Trajectory Compression Techniques

Previous algorithm for compressing GPS trajectories including Douglas-Peucker, Dead Reckoning and Uniform sampling each have serious limitations. These limitations are the main motivation for development of a new algorithm (SQUISH) that tries to overcome these limitations.

Uniform Sampling is a trivial compression strategy that simply takes every ith point in the trajectory. It therefore executes in linear time. However, the algorithm can easily miss sharp changes in speed and direction if the sampling rate is too large.

Douglas-Peucker is a popular algorithm for line generalization and is often used in cartographic and computer graphics applications. However, Douglas-Peucker is a batch algorithm. Most applications that require compressing of GPS trajectories cannot be implemented in batch-mode. This is because the data is arriving on a portable device as a data stream. Since portable devices have a limited amount of space for data, it is impractical to store the entire stream before it is compressed. Furthermore, real-time knowledge of tracked moving objects requires periodic updates of object locations.

Dead Reckoning has several useful advantages compared to Uniform Sampling and Dead Reckoning. First, the algorithm incorporates both spatial and temporal information when selecting points, thereby improving performance when evaluating sed and speed error. Also, execution time is linear to the number of trajectory points and the algorithm is able to run in online mode. The downside to Dead-Reckoning is the performance is only slightly better then other algorithms with respect to median synchronized Euclidean distance, and does not significantly improve the preservation of speed data.

The shortcomings of other algorithms motivated the development of SQUISH. This online algorithm demonstrates impressive performance when at least 10% of the original data points remain in the compression. The results will be described in detail in Section 4. The next section provides a detailed description of the SQUISH algorithm.

3. SQUISH METHOD

The main contribution of this paper is the introduction of a new algorithm call SQUISH for compressing GPS trajectory streams using a priority queue.

The algorithm requires one input parameter that defines the size of Buffer β. Initially all incoming points are simply placed in the buffer, until the buffer is full. Once the buffer is filled, any incoming point requires the removal of another point inside the buffer. The goal is to remove points that contain little or no information. Examples of trajectory segments that contain a high degree of redundancy include stationary points and points that are moving in a highly predictable constant speed and direction.

Prioritization is determined based on estimating the amount of synchronized Euclidean distance (sed) introduced into the compression if that point was removed from the compres-

Algorithm 1: SQUISH Method($\mathcal{S}$)

```
construct a fixed size buffer B
for GPS point p in continuous point stream S do
    insert p into B as the last point
    update the SED for the second last point in B
    if the buffer is full then
        find the point p_s in B with the smallest SED
        remove p_s from B
        update SED for two neighbor points of p_s in B
output B
```

Symbol	Meaning
$\mathcal{S}$	the continuous GPS point stream
$\mathcal{B}$	the buffer to accommodate the final compressed points
p	a GPS point in the stream $\mathcal{S}$
p_s	a GPS point in the buffer $\mathcal{B}$ that has the smallest SED

Table 2: Symbols and Their Meanings

sion. Optimally assigning priorities would be computationally expensive, since removal of any point in the algorithm would require reassigning priorities to every point in the buffer. Therefore, the prioritization algorithm uses local optimization instead of a more accurate global approach. For instance, in Figure 3, assume point P_3 was initially assigned the lowest priority since removing this point would result in the least amount of error introduced into the compression. However, removing point P_3 also increases the priority of its neighbors, P_2 and P_4. This is because the neighbors now contain some more important information that reflects, not only their spatio-temporal information, but also additional information from the deleted neighbor P_3. A heuristic is implemented to efficiently estimate the priority of the neighbor of a deleted point.

SQUISH identifies extreme points based on local estimation of sed error. Extreme point are defined as trajectory locations that exhibit distinct deviations of predicted behavior, such as sharp changes in speed and direction. These anomalies are often important to store for applications, such as the detection of erratic driving behavior, route changes and congestion.

Figure 4 captures three time segments of the SQUISH method in action. The dotted box depicts which points have currently been processed. The values next to each point show the current estimated priority of each point. The end point has a priority set to ∞ since ends points must remain in the compression and cannot be removed. The priority is set to the estimated sed error introduced into the compression if this point is removed. If the buffer becomes full, the point with the lowest estimated sed is removed when a new point is added into the buffer.

This small example has a buffer that can hold a maximum of four points, shown below each trajectory. The first four points are simply added to the buffer until the buffer is full, shown at time step 0. At time 1, a fifth point is processed. In order to store this incoming point, we store, then check if the buffer is full . If the buffer is full, some point needs to be removed from the buffer. To estimate the priority, a line

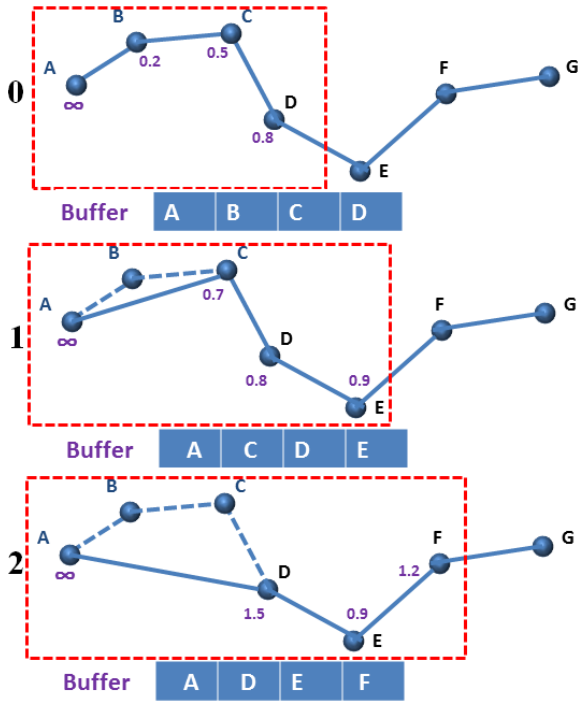

Figure 4: SQUISH estimates the lowest sed error and removes the point with is predicted to introduce the lowest amount of error into the compression

segment connecting the two adjacent neighbors is computed. For instance, a line segment AC is constructed and the sed from point B to line segment AC is computed. This process is carried out for all non-endpoints in the buffer. In the case of Figure 4, point B is removed since it has the lowest estimated sed error.

When a point is removed from the buffer, priorities of other points currently in the buffer are also affected. We estimate the upper bound maximum sed error of removing the two adjacent neighbors of the deleted point. This is computed by adding the sed error of the deleted point to the current sed error of the neighboring two points. In Figure 4, at time 1, point C has the sed changed from 0.5 to 0.7, by adding the priority of B to the current value at C. At time 2, point C is removed since it has the lowest sed, and hence highest priority. The estimated sed of point D is increased since the estimated sed of point C is added to the existing value of D. This process continues, as each new point in the GPS trajectory stream is added, the point with the lowest sed is removed.

The current implementation of SQUISH utilizes a fixed size buffer. This is useful when the amount of available storfge is predetermined, such as the amount of storage allocated on a mobile GPS collection system. However, modifications can be made to SQUISH that vary the buffer size dynamically based on global or local error metrics. For instance, a parameter could be fed to SQUISH that states a maximum sed error that is permitted in the compression. If the compression representation is about to surpass this maximum error, then the buffer size is allowed to grow in order to have a more accurate representation.

Algorithm 1, describes the SQUISH process. First, a fixed

size buffer β is initialized in which the size determines the number of points that will exist in the final compressed data. SQUISH iterates over all the points in the GPS trajectory stream. Each point p_i is added as the last point in the buffer β (line 3). This newly added point p allows for the calculation of the sed error for the second to last point p_{i-1}. The sed error of p_{i-1} is computed by measuring the amount of sed error introduced if p_{i-1} is removed from the compressed storage. If the buffer is full, then the point with the smallest estimated sed error is removed. Line 6 performs the check to find the point with the lowest sed error. If the data structure is implemeneted as a heap, than this find operation takes $O(\log \beta)$ time. One the point with the lowest estimated sed error is removed, the sed values of the two adjacent neighbors must be adjusted (line 8). SQUISH estimates the upper bound sed error of the two neighbors by adding the sed value of the deleted point to the values of the two neighbors p_{s-1} and p_{s+1}.

4. EVALUATION

4.1 Dataset Description

This study uses the Microsoft GeoLife [18, 19] dataset that consists of 165 users over a period of two years (from April 2007 to August 2009). Various transportation modes are included in the data set, including biking, shopping and work travel. Each trajectory represents a complete trip from destination to arrival location. The vast majority of the traces (about 95%) have a sampling rate of about 2 - 5 seconds or 5 - 10 meters, while a small fraction have more sparsely collected sampling rates. Most of the data collection occurred around Beijing, China, with a small number of traces including the United States and Europe.

Noise and inaccuracies are common in GPS data due to limitations of the GPS device, atmospheric conditions and positions of satellites. The location of the GPS device also determines the accuracy of the position. Entering a building or tunnel (the so-called urban canyon effect), can led to problems in obtaining an accurate GPS fix. The GeoLife data set contained noisy information and therefore was cleaned prior to the analysis performed in this paper. Removing noisy and grossly invalid points is important to understand the effectiveness of the algorithms without noisy information skewing the results. The data set was cleaned to extract long, complete trips that have little noise. A total of 1300 trips were extract from the GeoLife data set to facilitate a clean comparison of the different algorithms.

4.2 Performance Evaluation

Four algorithms were compared in this study. Three algorithms (Uniform Sampling, Douglas-Peucker and Dead Reckoning) were previously defined in the literature and used to gauge the effectiveness of the new algorithm SQUISH. This section compares the algorithms across multiple performance metrics including synchronized Euclidean distance, speed and the ranking of best performers.

SQUISH demonstrates the best results when the compression ratio is not large (e.g. at least 10% of the original points remain). The **compression ratio** is defined as the number of original points divided by the number of compressed points stored.

Figure 5A shows the average synchronized Euclidean distance (sed) ranking from a compression ratio of 5 to 30. SQUISH performs well for small compression ratios. For example, at a compression ratio of 5, in which 20% of the original trajectory points are stored, SQUISH provides roughly half the error rate of the second most effective algorithm Dead Reckoning and about 25% of the error rate of Uniform Sampling and Douglas-Peucker. Performance significantly deteriorates after compression ratio of 10, and SQUISH becomes the least effective after a compression ratio of 25.

For very high compression ratios, defined in this work as about 20, the performance of the SQUISH algorithm is quite poor. This is likely due to error propagation that stems from the weighting of points in the buffer. Points are ranked using a localized technique that estimates the amount of error introduced when removing a point. When a point is removed from the buffer, the remaining neighbor points are given a higher weight by adding the amount to error introduced by its neighbor. This technique is a good approximation; however, as more and more points are removed from the buffer, error propagation occurs. Enhancements are needed to this algorithm to decide how points are ranked to improve performance at high compression rations by limiting or preventing error propagation from occurring.

A ranking was performed that provides an ordered list for each analyzed trace to determine which algorithm is the best, second best, third best and worst performer. The ranking was estimated based on root mean square synchronized Euclidean distance for each trace in the Microsoft GeoLife dataset. A ranking of 1 for a particular trace illustrates that the algorithm had the lowest RMS sed error compared to the other three algorithms. Similarly, if an algorithm had a ranking of 4, the algorithm had the highest sed error compared to the other three algorithms.

Figure 5B illustrates the average ranking of the four algorithms over the entire GeoLife dataset and across various compression ratios. On small compression ratios (≤ 10) the SQUISH is shown to be the clear winner. In fact, for a compression ratio of 5, about 85% of the traces in the data set were best compressed using SQUISH. However, as the compression ratio increase, SQUISH eventually becomes the worst performer.

Another observation depicted in Figure 5 is that Douglas-Peucker is shown to perform poorly compared to the other three algorithms. This is due to the fact that Douglas-Peucker algorithm attempts to minimize the maximum perpendicular distance error, and does not include spatio-temporal information. Uniform Sampling, also does not use spatio-temporal information. However, very large sed values are often avoided since the amount of time between samples is constant and free from large gaps that can occur when using Douglas-Peucker.

In Figure 6A the median sed error is shown for the four algorithms. The median error removes outliers and therefore, best-case/worst-case compressions are not captured. Using median sed as the error metric, SQUISH is still shown to be the best performer for low compression ratios. However, the comparative difference between the four algorithms is not as drastic. Consistent with the results from the average sed error, Douglas-Peucker remains the worst performer over small compression ratios when using median error.

SQUISH demonstrated superior performance when measuring the amount of speed error. Figure 6B shows the average speed error for each algorithm over various compression ratios. The performance of each algorithm was fairly con-

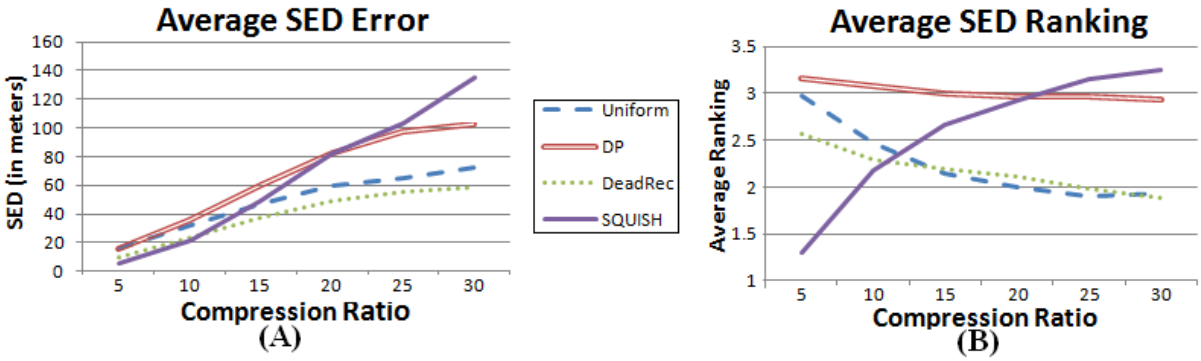

Figure 5: SQUISH method performs the best for small compression ratios. LEFT: Comparison of average synchronized Euclidean distance (sed) across four algorithms. RIGHT: Average ranking of the algorithms. A ranking of 1 indicates that the algorithm had the lowest sed error. Similarly, a ranking of 4 indicates that the algorithm had the highest sed error. For compression ratios less than 10, SQUISH clearly has the best overall ranking. This difference between SQUISH average sed results and overall ranking is due to small numbers of outliers that skew the overall average.

sistent. The SQUISH method consistently had the smallest amount of speed error, followed by Dead Reckoning, Douglas-Peucker and Uniform Sampling respectively.

When comparing speed performance, SQUISH and Dead Reckoning were the best algorithms because temporal information is used by the algorithms in deciding which points to select and which points to remove from the compressed representation. Since the temporal information is essential to preserving velocity, this likely influenced the algorithms' performance.

The performance of each algorithm on synchronized Euclidean distance and speed is poorly correlated. For instance, Douglas-Peucker was comparable to the SQUISH method for compression ratios 10 through 20 for average sed, but the median speed error was significantly worse than the sed error over the same compression ratios.

The cumulative distribution function is shown in Figure 7 for the four algorithms. The x-axis indicates the amount of sed error in kilometers, while the y-axis represents the percentile of the distribution with that amount of error. A higher percentile corresponds to a higher error. This figure shows the distribution of sed error at a compression ratio of 5. The SQUISH method is shown to outperform other algorithms by having a distribution shifted further left which represents that the worst-case error is less using SQUISH than other methods. Douglas-Peucker is the second best, followed by Dead Reckoning and Uniform Sampling.

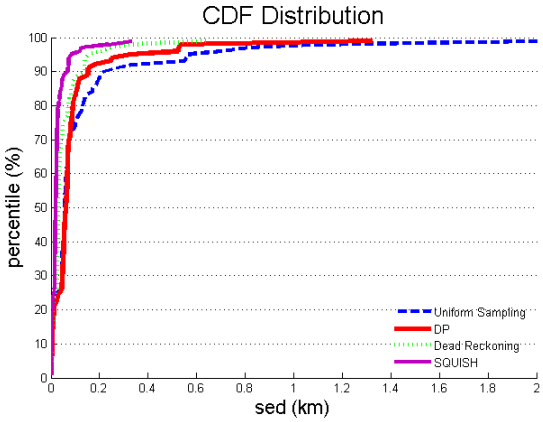

Figure 7: CDF SED Distribution

5. CONCLUSION

The SQUISH algorithm offers several advantages over previous algorithms reported in the literature. The main advantages are improved performance with respect to synchronized Euclidean distance (sed) when the compression ration is 10 or smaller using an online method. The distribution of error demonstrates that SQUISH has smaller sed error rates under worst-case scenarios. Additionally, speed information is well preserved using the SQUISH method, which consistently out-performs other compression algorithms even when the trajectories are substantially compressed (i.e. 20-25% of the original points are stored).

Future work is needed to minimize the algorithm's drawbacks, such as using a more accurate heuristic for estimating the sed error of points in the queue. The local estimation of the priority of each remaining point in the buffer is currently incapable of handling large compression rates. The performance of SQUISH when 10% or more of the original points remain indicates that SQUISH could be used as a preprocesing algorithm for aggressive compression. Experimental results are needed to validate whether SQUISH is effective as a preprocessing algorithm. Additional recommendations for future work include determining the effectiveness of compression on common spatial applications such as traffic flow modeling, identification of congestion bottlenecks and identification of speeding violation hot-spots.

6. ACKNOWLEDGMENTS

This work is supported in part by the Research and Innovative Technology Administration of the U.S. Department of Transportation through the Region 2 - University Transportation Research Centers Program and the University at Albany through the Faculty Research Awards Program (FRAP - A). We thank the anonymous reviewers for helpful suggestions.

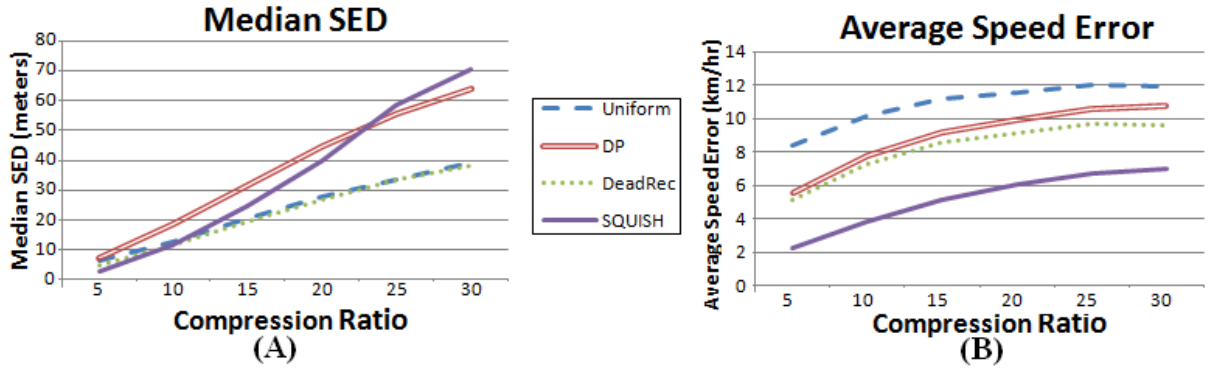

Figure 6: LEFT: Median synchronized Euclidean distance error across different compression ratios. RIGHT: The average speed error for various compression rates. SQUISH approach works well in preserving speed information, but has a high sed error on high compression ratios

7. REFERENCES

[1] M. Abdelguerfi, J. Givaudan, K. Shaw, and R. Ladner. The 2-3tr-tree, a trajectory-oriented index structure for fully envolving valid-time spatio-temporal datasets. In *10th ACM-GIS*, pages 29–34. ACM Press, 2002.

[2] P. K. Agarwal, L. J. Guibas, H. Edelsbrunner, J. Erickson, M. Isard, S. HarPeled, J. Hershberger, C. Jensen, and L. Kavraki. Algorithmic issues in modeling motion. *ACM Computing Surveys*, 34:550–572, 2002.

[3] 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.

[4] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424 – 436, 1993.

[5] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi. Trajectory pattern mining. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 330–339, New York, NY, USA, 2007. ACM.

[6] S. P. Greaves and M. A. Figliozzi. Commerical vehicle tour data collection using passive gps technology: Issues and potential applications. *Transportation Research Record*, 2049:158–166, 2008.

[7] J. Hershberger and J. Snoeyink. Speeding up the douglas-peucker line simplification algorithm, 1992.

[8] C. Jones and J. Sedor. Improving the reliability of freight travel. *Public Roads*, 70(1), 2006.

[9] E. McCormack and M. E. Hallenbeck. Its devices used to collect truck data for performance benchmarks. *Transportation Research Record*, 1957:43–50, 2006.

[10] N. Meratnia and R. A. d. By. *Spatiotemportal Compression Techniques for Moving Point Objects*, volume 2992. Springer Berlin / Heidelberg, 2004.

[11] J. Muckell, Q. Cao, P. Mackenzie, D. Messier, and J. Salvo. Toward an intelligent brokerage platform: Mining backhaul opportunities in telematics data. *Transportation Research Record*, 2097:1–8, 2009.

[12] J. Muckell, J.-H. Hwang, C. T. Lawson, and S. S. Ravi. Algorithms for compressing gps trajectory data: an empirical evaluation. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 402–405, New York, NY, USA, 2010. ACM.

[13] M. Potamias, K. Patroumpas, and T. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*, pages 275–284, 2006.

[14] M. Prior-Jones. *Satellite Communications Systems Buyer's Guide*. British Antarctic Survey, 2008.

[15] J. F. Srour and D. Newton. Freight-specific data derived from intelligent transportation systems: Potential uses in planning freight improvement projects. *Transportation Research Record*, 1957:66–74, 2006.

[16] G. Trajcevski, H. Cao, P. Scheuermanny, O. Wolfsonz, and D. Vaccaro. 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 '06, pages 19–26, New York, NY, USA, 2006. ACM.

[17] P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, pages 75–84, Washington, DC, USA, 1975. IEEE Computer Society.

[18] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma. Understanding mobility based on gps data. In *Proceedings of the 10th international conference on Ubiquitous computing*, UbiComp '08, pages 312–321, New York, NY, USA, 2008. ACM.

[19] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 791–800, New York, NY, USA, 2009. ACM.

[20] H. Zhu, J. Su, and O. H. Ibarra. *Trajectory queries and octagons in moving object databases*. ACM Press, 2002.