IMPROVEMENT OF TD-TR ALGORITHM FOR SIMPLIFYING GPS TRAJECTORY DATA

BY

MR. KANASUAN HANSUDDHISUNTORN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE (ENGINEERING AND TECHNOLOGY)

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THAMMASAT UNIVERSITY

ACADEMIC YEAR 2019

IMPROVEMENT OF TD-TR ALGORITHM FOR SIMPLIFYING GPS TRAJECTORY DATA

BY

MR. KANASUAN HANSUDDHISUNTORN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE (ENGINEERING AND TECHNOLOGY)

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THAMMASAT UNIVERSITY

ACADEMIC YEAR 2019

THAMMASAT UNIVERSITY
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THESIS

BY

MR. KANASUAN HANSUDDHISUNTORN

ENTITLED

IMPROVEMENT OF TD-TR ALGORITHM FOR SIMPLIFYING GPS TRAJECTORY DATA

was approved as partial fulfillment of the requirements for
the degree of Master of Science (Engineering and Technology)

on December 9, 2019

Chairperson _____
(Assistant Professor Apichon Witayangkurn, Ph.D.)

Member and Advisor _____
(Assistant Professor Teerayut Horanont, Ph.D.)

Member _____
(Virach Sornlertlamvanich, D.Eng.)

Director _____
(Professor Pruettha Nanakorn, D.Eng.)

Thesis Title	IMPROVEMENT OF TD-TR ALGORITHM FOR SIMPLIFYING GPS TRAJECTORY DATA
Author	Mr. Kanasuan Hansuddhisuntorn
Degree	Master of Science (Engineering and Technology)
Faculty/University	Sirindhorn International Institute of Technology/ Thammasat University
Thesis Advisor	Assistant Professor Teerayut Horanont, Ph.D.
Academic Years	2019

ABSTRACT

Over the past decades, the amounts of GPS-enabled devices sold has drastically increased exponentially, with the rapid growth of modern technologies and manufacturing cost. The GPS-enabled devices such as smart phones and wearable devices are capturing massive amounts of users' spatial and temporal information. Due to the increasing number of GPS-enabled devices, the volume of spatial and temporal information recording the footprint of a moving device has increased dramatically. The massive amounts of trajectory data could easily exceed the existing available data storage, which leads to three major challenges: storing, transmitting and visualizing the data.

While dealing with these massive amounts of trajectory data, an effective compression mechanism is needed. The generic compression technique is called trajectory simplification, which results in a corresponding approximation of the initial path and aims to minimize the minimize information loss while preserving the quality of the information under a specific error threshold. Several of trajectory simplification algorithms have been proposed to fulfil these demands.

In this study, an improved algorithm for top-down time-ratio (TD-TR) called top-down time-ratio Reduce (TD-TR Reduce) is proposed. The algorithms were

evaluated using several parameters, such as compression times and errors arising from trajectory data simplifications. The proposed TD-TR Reduce simplification method is applied on Geolife GPS trajectory dataset. The results of trajectory simplification are reported and compared with that from traditional TD-TR algorithm. The effectiveness of simplification is evaluated.

The results of the simulation reveal that the proposed method can achieve an attractive trade-off between the compression rate and the simplification error while having shorter compression time.

Keywords: GPS, Trajectory Data, Trajectory simplification

ACKNOWLEDGEMENTS

First of all, I would like to express my special gratitude to my advisor Assistant Professor Dr. Teerayut Horanont for his teaching, guidance, and especially his continuous support during my master's program.

Second, I would like to express my sincere gratitude to the members of my Review Committee: Dr. Virach Sornlertlamvanich and Assistant Professor Dr. Apichon Witayangkurn for their kind participation and comments.

Third, I would like to express my gratitude to Sirindhorn International Institute of Technology (SIIT), Thammasat University, for the Excellent Foreign Students (EFS) scholarship program. I would also like to thank the staff of the SIIT and the Secretary of the School of Information, Computer and Communication Technology who are helping with the paperwork and providing guidance to my life during my studies.

Finally, I would like to express my deep gratitude to my beloved family, especially to my father and mother, for always standing beside me and supporting me.

Mr. Kanasuan Hansuddhisuntorn

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

Symbols/Abbreviations	Terms
SIIT	Sirindhorn International Institute of Technology
TU	Thammasat University
DP	Douglas-Peucker
TD-TR	Top-down time-ratio
MRPA	Multiresolution Polygonal Approximation
PED	Perpendicular Euclidean Distance
SED	Synchronized Euclidean Distance
ASED	Average Synchronized Euclidean Distance
TDDR	Trajectory distance reduction ratio

CHAPTER 1
INTRODUCTION

1.1 Background of the study

Over the past decade, the number of GPS-enabled devices has increased significantly as GPS-enabled portable devices become easily available (Muckell et al.,2013). Due to the increasing number of GPS-enabled devices, such as mobile phones or in-car navigation systems, the trajectory data has increased dramatically. GPS trajectory data contain spatial and temporal information that records the digital footprint of a moving subject's device activity, such as consecutive points in the trajectory that may indicate certain activities (e.g. finding parking lots) or routines (e.g. sending children to school every morning) or changes in the mode of transport (e.g. Switching from walking to public bus). The massive amounts of trajectory data could easily exceed the existing available data storage. For example, a calculation due to Meratnia and de By (2004) shows that without any data compression, storing a trajectory data of 400 objects per day at an interval of 10 seconds requires a storage capacity of 100 Mb. This creates several problems in location-based services: (1) The amount of storage spaces required to store an enormous amount of GPS trajectory. (2) The amount of bandwidth and other overhead generated during the process of data transmission. (3) The amount of computational power required to visualize the GPS trajectory data. To solve these problems, a scalable data compression method for trajectories is required.

In order to reduce the amount of trajectory data, trajectory simplification has been adopted. Many trajectory simplifications types have been introduced in the past which can be divided into various methods of categorization, lossless compression and lossy compression or online compression and offline compression. The general of trajectory simplification is to remove some trajectory points while retaining an acceptable degree of error.

1.2 Problem statement, research aim and research objectives

As we have seen, the massive amount of trajectory data could exceed the existing storage capacity. While dealing with these massive amounts of trajectory data, an effective mechanism for compression is needed. Due to the computational overhead of the existing algorithms, we define our problem as a trajectory simplification problem. This study, therefore, aims to propose an improvement of trajectory simplification algorithm based on an existing current state of the art algorithm. To achieve the research aim, the following objectives are defined:

- Introduces an offline lossy trajectory simplification algorithm, which results in a shorter compression time against the current state of the art compression algorithms.
- Perform a performance evaluation among the popular trajectory simplification algorithms: Douglas-Peucker, Top-down time-ratio and Multiresolution polygonal approximation.
- Suggestions on how to develop future algorithms to performance improvement.

1.3 Thesis organization

This thesis consists of six chapters. The main content of each chapter is described as follows.

- Chapter 1 introduces background of the study, including the characteristics of the GPS trajectory. The chapter presents the problem statement, research aim, and research objectives at the end of the chapter
- Chapter 2 reviews literature related to trajectory simplification algorithm.
- Chapter 3 introduce definition, metrics and discussion of the ideal outcome as well as the introduction of average time-synchronized euclidean.
- Chapter 4 explains the proposed approach and feature point extraction model.
- Chapter 5 explains dataset, experiment setting and the comparison of trajectory simplification algorithm. Experiment results are also detailed and discussed in this chapter.
- Chapter 6 summarizes major findings, limitations, and recommendations for future research.

CHAPTER 2
REVIEW OF LITERATURE

2.1 Introduction

Several algorithms have been proposed to simplify trajectory data, different algorithms use different approaches to find a similar trajectory with fewer points. Trajectory compression can be categorized as online and offline compression or lossless and loss compression, depending on the various categorization methods. The advantage of online compression is that it supports real-time applications and can compress trajectory data while collecting upcoming trajectory points. Only after all points have been obtained from the input trajectory, offline compression can be performed. However, offline compression usually has smaller errors compare to online compression. Lossless compression enables the original data to be reconstructed without loss of information, while Lossy compression is not possible. The main advantage of lossy compression is that it can significantly reduce the trajectory size while maintaining reasonable error tolerances. The four algorithms in this paper were compared, three of which have already been mentioned in the literature.

2.2 Douglas-Peucker

Douglas-Peuker (DP) algorithm by Douglas and Peucker (1973), compresses trajectory data by recursively divides the trajectory to decide which points should be retained according to user-defined Perpendicular Euclidean Distance (PED) threshold. This algorithm begins with a rough simplification of the edge, which is the one edge connecting the initial and the last vertices of the original polyline. Once the recursion is done, a new performance curve consisting of only those marked as retained will be generated as demonstrated in Figure 2.1.

2.3 Top-down time-ratio

Top-down time-ratio (Meratnia and de By 2004) is an extension of the Douglas-Peucker. The difference between them is that where the Douglas-Peuker algorithm uses the Euclidean distance as an error metric, the TD-TR algorithm uses a time-synchronous Euclidean distance called Synchronized Euclidean Distance to overcome the limitation of ignoring temporal data. Figure 2.2 illustrate how TD-TR algorithm iteratively simplifies a line.

2.4 Multiresolution polygonal approximation

Multiresolution polygonal approximation or MRPA algorithm (Minjie et al.,2012) is proposed to compress trajectories in O(N) computational space, where a new error metric called an integral square synchronous Euclidean distance (ISSD) was introduced. The MRPA algorithm uses the bottom–up multiresolution approach. The proposed method made substantial progress in the real-time application solution of the GPS Trajectory Simplification problem.

Algorithm	Time Complexity	Error Criterion	Offline
DP	$O(n^2)$	PED	Yes
TD-TR	$O(n^2)$	SED	Yes
MRPA	$O(n^2/M)$	ISSD	Yes

Table 2.1: Simplification algorithm summary

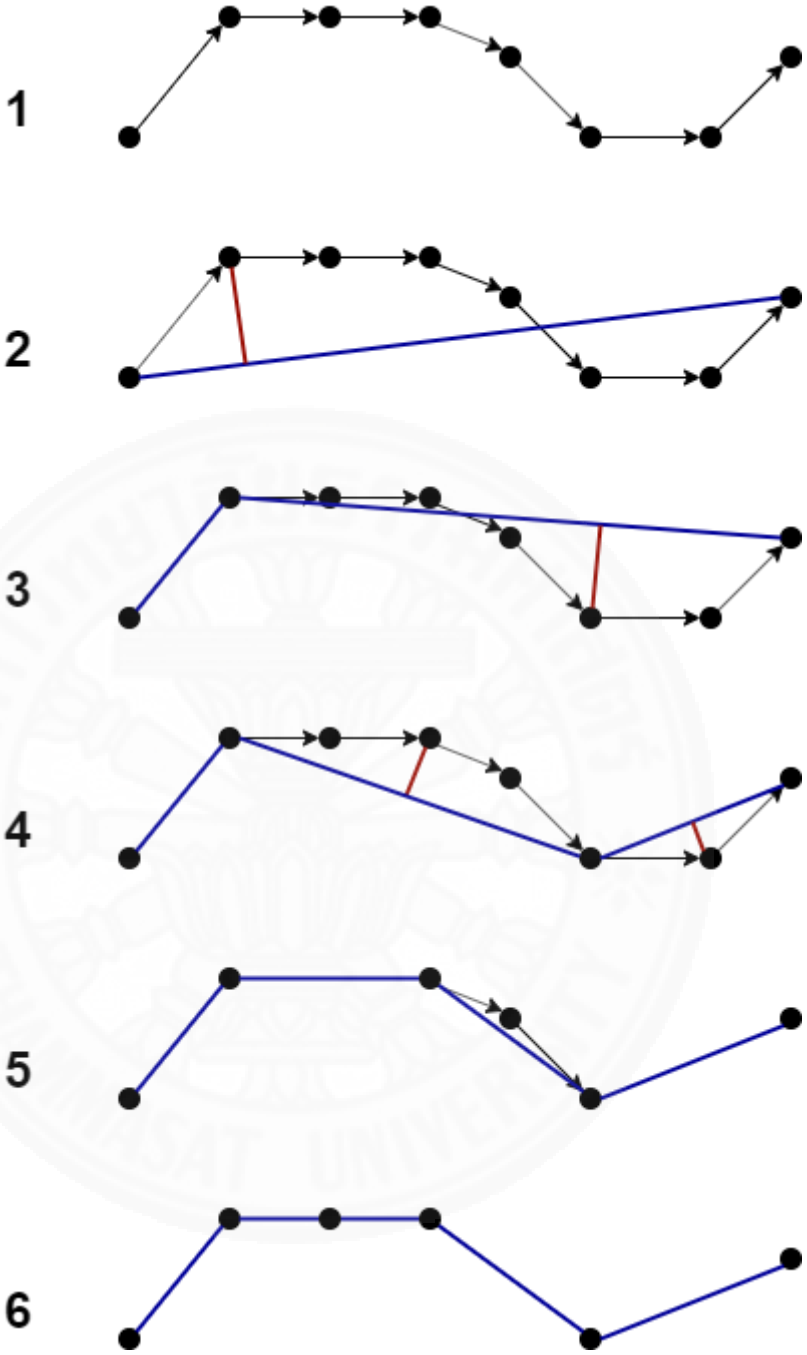

Figure 2.1 Illustration of Douglas-Peucker algorithm.

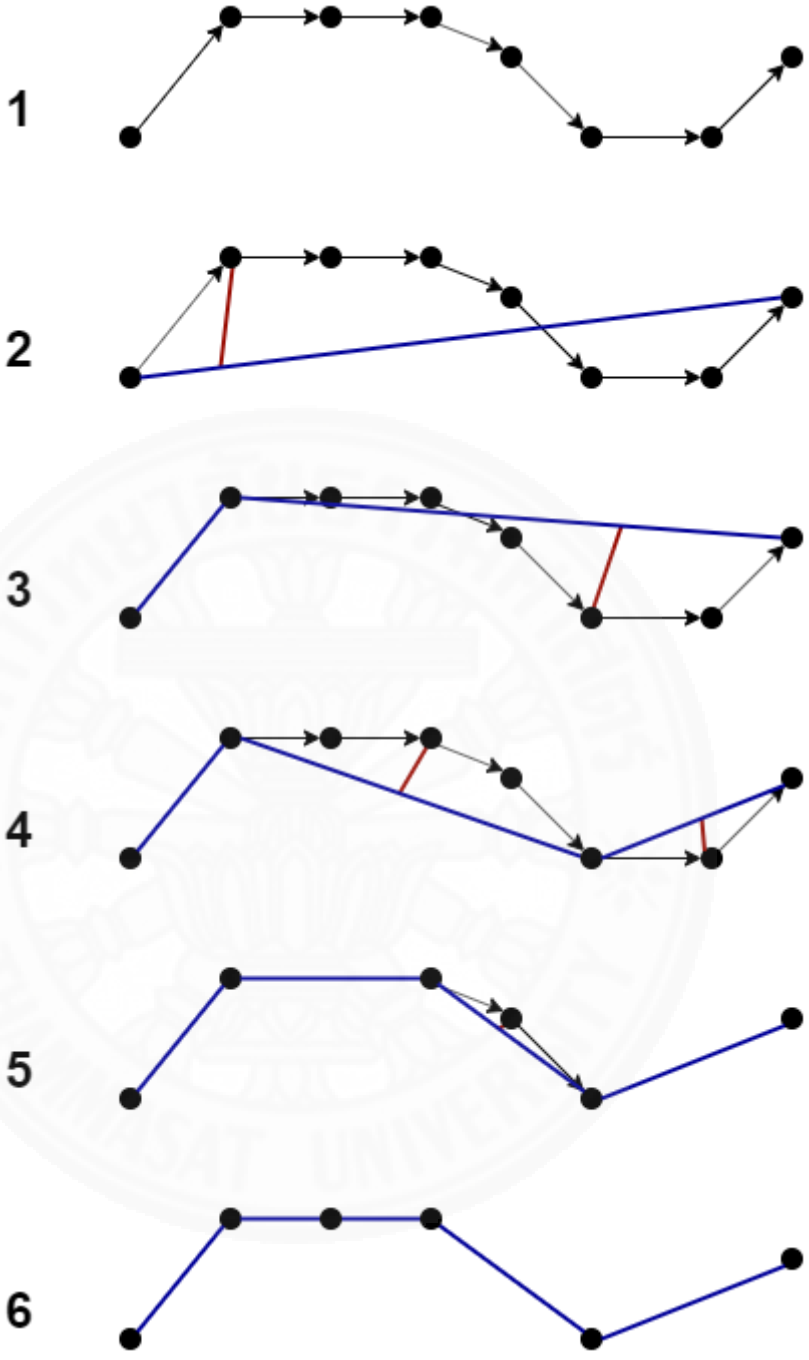

Figure 2.2 Illustration of TD-TR algorithm.

CHAPTER 3
DEFINITION AND METRICS

3.1 Introduction

This section describes both the necessary definition and the metrics for evaluating the simplification algorithm. First, we introduce a few terms used in this thesis, and then we define our metric.

3.2 Definition

3.2.1 GPS Point

A GPS point P_i is a tuple $P_i(x_i, y_i, t_i)$ that contain longitude x, latitude y and timestamp t of the *i*-th point, where $i = \{1,2, \dots, n\}$ and *n* represent the number of GPS points in a trajectory.

3.2.2 Trajectory

A trajectory is a temporal ordered sequence of points denoted as $T = \{P_1, \dots, P_n\}$, where $|T| = n$ represents the size of trajectory T.

3.2.3 Simplified trajectory

Given an original trajectory $T = \{P_1, \dots, P_n\}$, a simplified Trajectory $T' \subseteq T$ is a subset of the original trajectory T and can be express as $T' = \{P_{s_1}, \dots, P_{s_m}\}$ where $m \leq n$ and $1 = s_1 < \cdots < s_m = n$

3.2.4 Spatial distance

Spatial distance is the length of the straight line connected between two points location P_a and $P_b$denoted by *DISTANCE* (P_a, P_b) and can be calculated as follows:

$$DISTANCE(P_a, P_b) = 6{,}371.00 \times \sqrt{((long_b - long_a) \times COS(\frac{lat_a + lat_b}{2}))^2 + (lat_b - lat_a)^2}$$

3.3 Error metrics

3.3.1 Synchronized Euclidean Distance

Synchronized Euclidean Distance (SED) is the distance between the actual point $P_k(x_k, y_k, t_k)$ and its synchronized point $P'_k(x'_k, y'_k, t'_k)$ created by two points P_s and P_e at identical time stamps (see Figure 3.1) and can be calculated as follows:

$$SED(P_k) = \sqrt{(x_k - x'_k)^2 + (y_k - y'_k)^2}$$

where

$$x'_k = x_s + \frac{x_e - x_s}{t_e - t_s}(t_k - t_s)$$
$$y'_k = y_s + \frac{y_e - y_s}{t_e - t_s}(t_k - t_s)$$

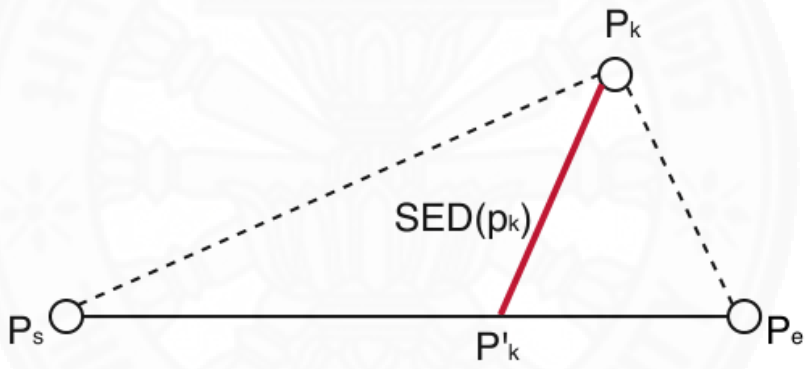

Figure 3.1 Illustration of Synchronized Euclidean Distance between P_s and P_e

3.3.2 Trajectory Distance Reduction Ratio

Trajectory distance reduction ratio (TDDR) is the accumulated travel distance ratio of the simplified trajectory T'versus its original trajectory T and can be calculated as follows:

$$TDRR(T, T') = 1 - \frac{TDD(T)}{TDD(T')}$$

where

$$TDD(T_i) = \sum_{i=1}^{|T|-1} DISTANCE(P_i, P_{i+1})$$

3.4 Performance metrics

3.4.1 Compression ratio

Compression ratio (CR) is defined as the size of the simplified trajectory T' versus its original trajectory T and can be calculated as follows:

$$CR(T, T') = 1 - \frac{|T|}{|T'|}$$

3.4.2 Compression time

Compression time (CT) is the amount of time taken for a trajectory to be simplified.

3.5 Discussion

The trajectory-simplifying algorithm's efficiency is defined as the combination of error metrics and performance metrics. For the further improvement of error metrics, in order to measure the average time-synchronized euclidean distance between the original trajectory T and its simplified trajectory T', we introduce Average Synchronized Euclidean distances (see Figure 3.2). Given Trajectory $T = \{P_1, P_2, P_3, P_4, P_5, P_6\}$ and its simplified trajectory $T' = \{P_1, P_4, P_6\}$. ASED is calculated as $(d_1 + d_2 + d_3)/2$. The ideal simplified trajectory should be highly compressed with minimum compression time, TDRR and ASED.

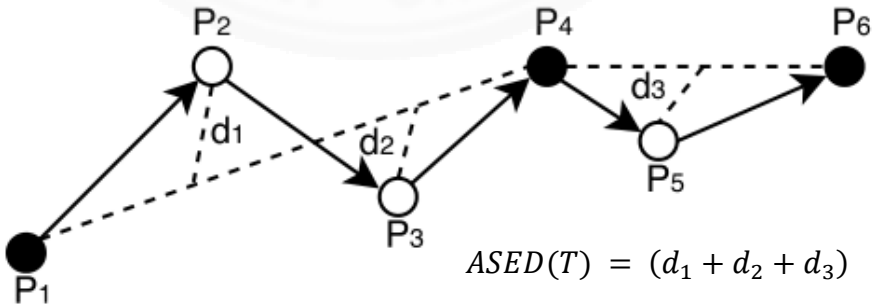

Figure 3.2 Example of calculating ASED.

CHAPTER 4
PROPOSED APPROACH

4.1 General overview

In this section we present our feature point extraction mechanism followed by our proposed algorithm TD-TR Reduce and trajectory simplification parameter determination.

4.2 Feature point extraction

Some GPS tracking point is redundant in some applications. To retain only the important part where some events occur (e.g. travel mode transition), we proposed a feature point extraction model with a focus reducing the trajectory data based on several of the movement characteristics given as follows.

4.2.1 Movement speed

In the transition mode, node movement speeds typically change significantly (Zheng et al.,2010) (see Figure 4.1), this includes walking, cycling, driving vehicles or taking the train. As shown in Figure 4.2, The feature node is the place where the node changes transportation mode from driving to walking. To detect a feature point caused by switching between transportation modes. First, we specify the difference of movement speed as $\Delta_{i+1} = |SP_{i+1} - SP_i|$, in which SP_i is an average speed in the line segment $\overline{P_{i,i+1}}$. When Δ_{i+1} exceeds a certain threshold, P_{i+1} is kept as a feature point. Because speed changes are usually caused by switches between travel modes. We propose the usage of the standard deviation of movement speeds as the speed threshold since the standard deviation can indicate changes in velocity. The speed threshold SP_{th} is expressed as

$$SP_{th} = \sqrt{\frac{\sum_{i=1}^{n-1}\left(SP_i - \overline{SP}\right)^2}{n-1}}$$

where

$$\overline{SP} = \sum_{i=1}^{n-1} \frac{SP_i}{n-1}]$$

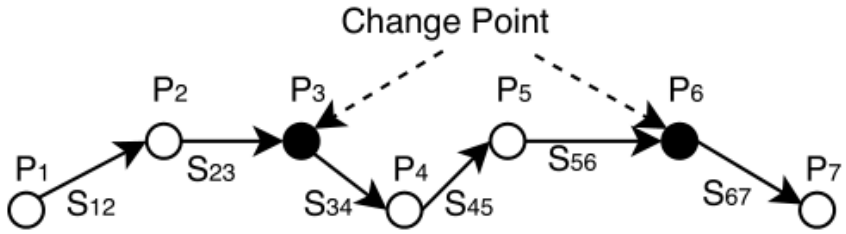

Figure 4.1 Movement speed change point

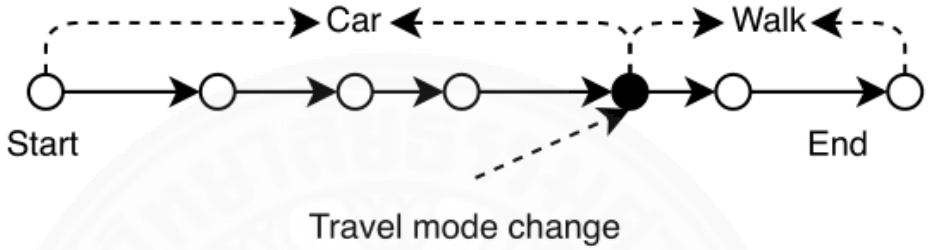

Figure 4.2 Transition of transportation mode

4.2.2 Heading

The heading changes accordingly to the current modes of transport. For example, when the mode of transport is walking rather than another mode of transport, the heading will change very often. To detect a feature point caused by a significant change in the node heading. δ_{i+1} was defined as the heading different between θ_i and θ_{i+1} and can be calculated as follows:

$$\delta_{i+1} = \begin{cases} 360 - |\theta_{i+1} - \theta_i|, & if\ |\theta_{i+1} - \theta_i| > 180 \\ |\theta_{i+1} - \theta_i|, & otherwise \end{cases}$$

where θ_i represents the current heading on the line segment $\overline{P_{i,i+1}}$, that can be calculated using the haversine formula. If δ_{i+1} exceeds a certain δ_{th}, P_{i+1} is kept as a feature point.

4.2.3 Skip threshold

When the node travels in a straight line at a constant speed (e.g. highway, tollway), the entire point will be ignored from the two-feature point extraction method mentioned above. In order to prevent the above scenario from occurring, we introduce

the skipping threshold $skip_{th}$. Figure 4.3 show the illustration of skip threshold. When the number of points that are ignored from the extraction method of a feature point above reaches the skipping threshold, which is 2 in this case, the point is kept as a feature point.

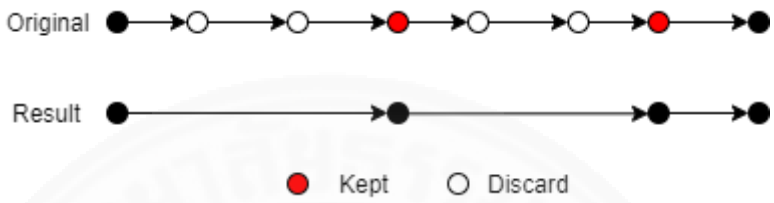

Figure 4.3 Illustration of feature point extraction when $skip_{th}$= 2

4.3 TD-TR Reduce

In order to reduce the compression time of the current TD-TR algorithm (see Figure 4.4), we propose a TD-TR Reduce algorithm to simplify the trajectory by performing a traditional TD-TR algorithm on a set of extracted feature points. The following procedure is provided in algorithm 1 below:

1. Add the first from T to the feature point array $T_{feature}$ (line 3)
2. Calculate the standard deviation of speed in T and set it as the speed threshold SP_{th} (line 4)
3. Starting from $i = 1$, iteratively add point p_{i+1} to the feature point array $T_{feature}$ and set skip parameter back to zero if the movement speed different Δ_{i+1} is greater than or equal to speed threshold SP_{th} or the heading different δ_{i+1} is greater than or equal to heading threshold δ_{th} (lines 5-7)
4. If the *n* point was not added to the feature point array $T_{feature}$, increase *skip* value by one (line 10)
5. If the *skip* value reaches the certain skip threshold $skip_{th}$, add point p_{i+1} to the feature point array $T_{feature}$ and set *skip* parameter back to zero (lines 12-15)

6. Add the last points in T to the feature point array $T_{feature}$ (line 18)
7. Perform TD-TR algorithm on feature points array $T_{feature}$ and stored it as a simplified trajectory T'. (line 19)
8. Finally, the simplified trajectory T' is returned. (line 20)

4.4 Trajectory-Simplification parameter determination

The rest of our approach problems are to evaluate appropriate parameters for skipping threshold $skip_{th}$ and heading threshold δ_{th} for TD-TR Reduce, respectively. If we reduce the number of $skip_{th}$, the resulting number of points in the feature point array will be lower, resulting in lower total compression time. In the opposite way, if the number of $skip_{th}$ was reduced, the resulting simplified trajectory will have higher ASED error and TDDR, and vice versa. Same as the heading threshold δ_{th},if we lower the number of δ_{th}, the resulting number of points in the feature point array will lower the overall compression time. In Section 5.3, in order to determine a proper parameter, we will conduct an experiment on a real-world dataset to study the outcome.

Algorithm 1 TD-TR Reduce algorithm

Input: T = $\{p_1,...,p_n\}$, heading threshold δ_{th}, error threshold ε, skip threshold $skip_{th}$

Output: Simplified Trajectory T'

1: $T_{feature} = []$
2: $skip = 0$, $i = 1$
3: $T_{feature} = T_{feature}$ **APPEND** p_1
4: $SP_{th} \leftarrow$ SD of speed in T
5: **while** $i < n - 1$ **do**
6: **if** $\mid \triangle_{i+1} \mid \geq SP_{th}$ **or** $\delta_{i+1} \geq \delta_{th}$ **then**
7: $T_{feature} = T_{feature}$ **APPEND** p_{i+1}
8: $skip = 0$
9: **else**
10: $skip = skip + 1$
11: **end if**
12: **if** $skip == skip_{th}$ **then**
13: $T_{feature} = T_{feature}$ **APPEND** p_{i+1}
14: $skip = 0$
15: **end if**
16: $i = i + 1$
17: **end while**
18: $T_{feature} = T_{feature}$ **APPEND** p_n
19: $T' = TDTR(T_{feature}, \varepsilon)$
20: **return** T'

Figure 4.4 TD-TR Reduce algorithm

CHAPTER 5
EXPERIMENTAL RESULTS AND DISCUSSION

5.1 General overview

This section introduces a dataset and then our proposed algorithms are evaluated based on three aspects: compression ratio, compression time and error metrics. Finally, we discuss the results and summarize the performance of the proposed algorithms. Three algorithms (DP, TD-TR, TD-TR Reduce) were written in Python, while MRPA was written in Matlab. The experiment is conducted on Windows 10 with 4 CPU cores (Intel i7-7700K with 4.20GHz) and 32 GB RAM.

5.2 Data

The Geolife dataset was collected by 182 participants in the (Microsoft research Asia) Geolife project (Zheng et al.,2010) for five years (from April 2007 to August 2012). different transport modes, including biking, walking and traveling, are included in the data set. most of the data collection has taken place in China, Beijing. more than 90% of trajectories are collected in a dense format, e.g. every 1 to 5 seconds or every 5 to 10 meters per point. the data set was cleaned to remove trajectories with high noise such as large jumps in time and space.

5.3 Experiment settings

For this simulation, three trajectories are chosen to observe the effect of $skip_{th}$ and δ_{th} on the number of remaining points. Figures 5.1-5.3 show the number of stop points under different heading threshold and skip threshold. The details of each trajectory are shown in Table 5.1. From Figures 5.1-5.3, the curve decreases dramatically when $skip_{th}$ increases from 2 to 3 and from 3 to 4. The curve starts to change slowly when $skip_{th} \geq 5$, so we set $skip_{th} = 5$. On the other hand, the number of remaining points decreases significantly when δ_{th}change from 45° to 60°and starts to change constantly when $\delta_{th} \geq 60°$, so we set $\delta_{th}$to 60°.

Trajectory ID	# of Points	Start	Stop
1	4,164	2009-02-20 04:01:36	2009-02-20 14:51:36
2	1,937	2008-10-28 23:51:59	2008-10-29 11:25:00
3	838	2009-02-24 12:16:55	2009-02-24 13:35:55

Table 5.1: Trajectory details

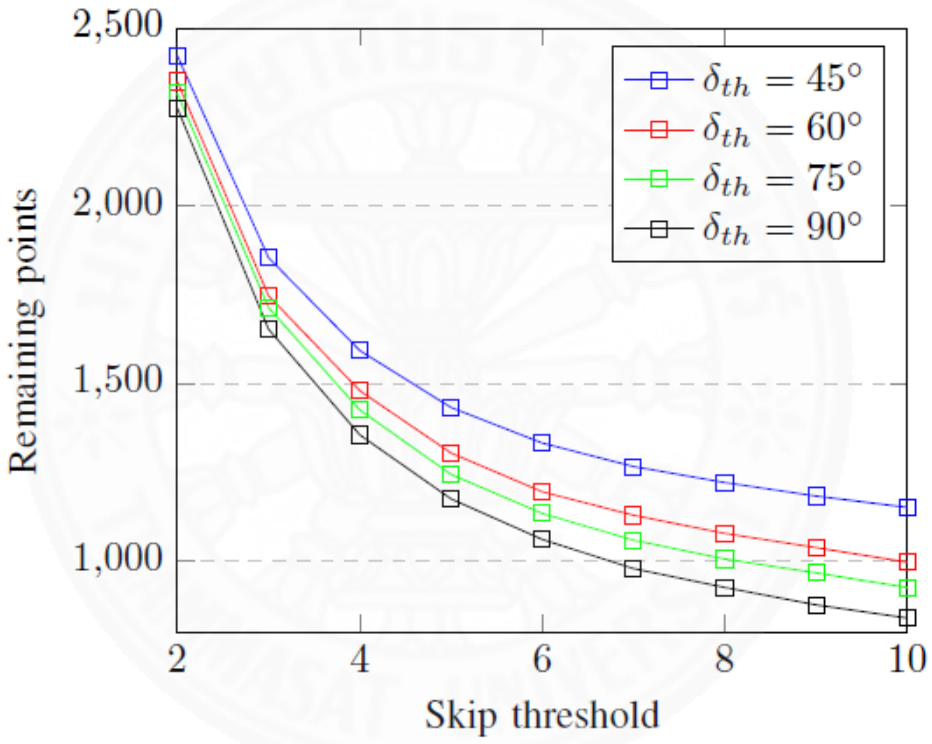

Figure 5.2 Trajectory one

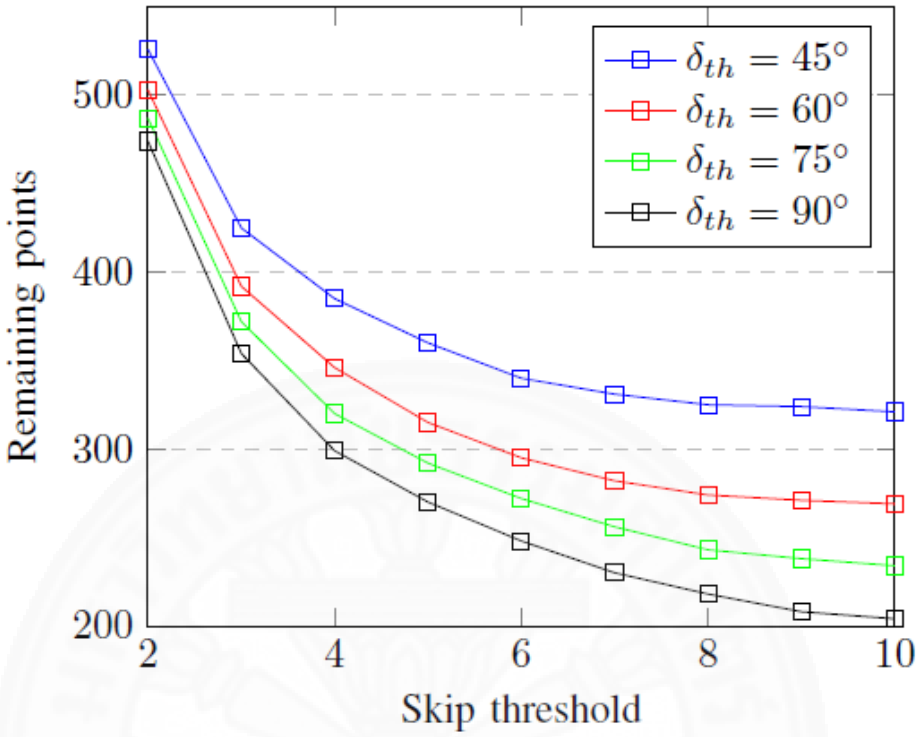

Figure 5.2 Trajectory two

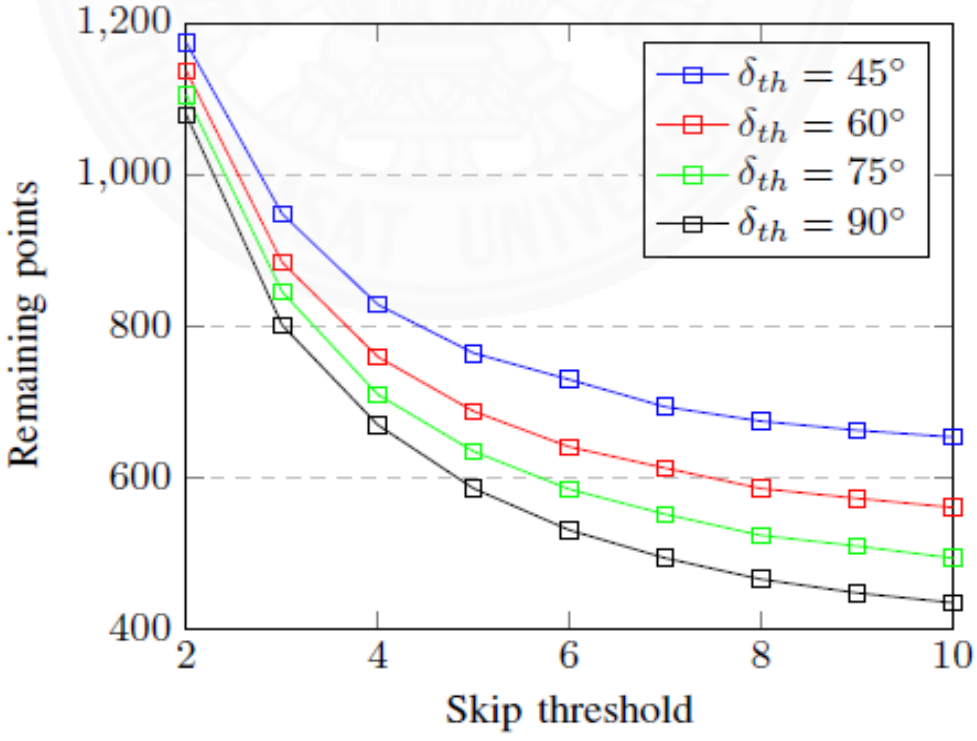

Figure 5.3 Trajectory three

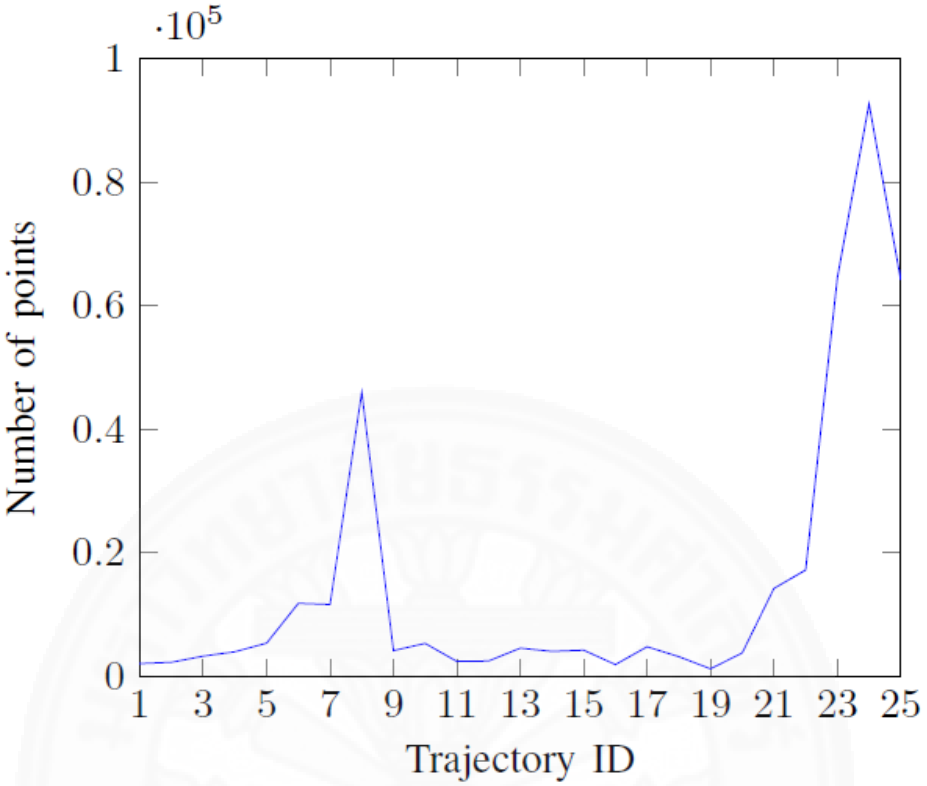

Figure 5.4 Trajectory details

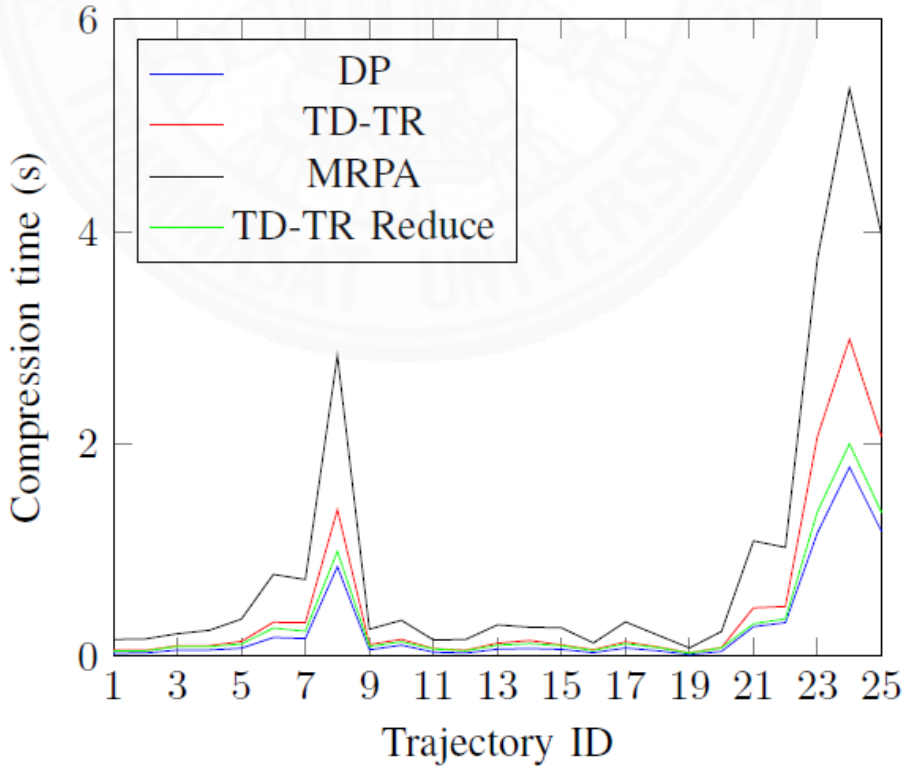

Figure 5.5 Compression time

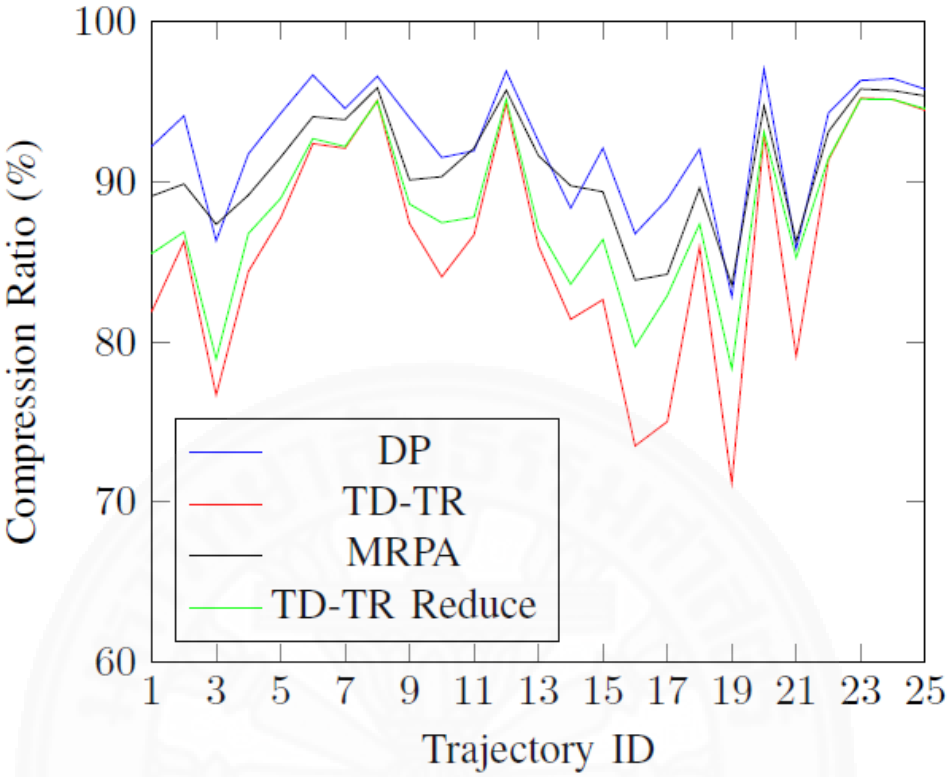

Figure 5.6 Compression ratio

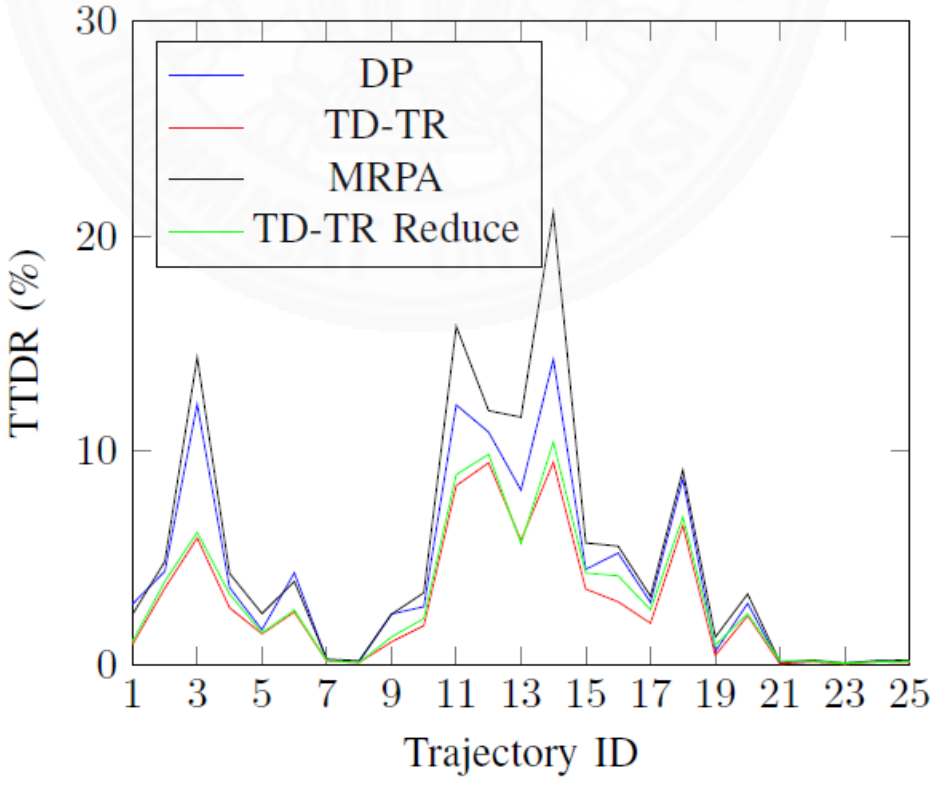

Figure 5.7 Trajectory distance reduction ratio (TDDR)

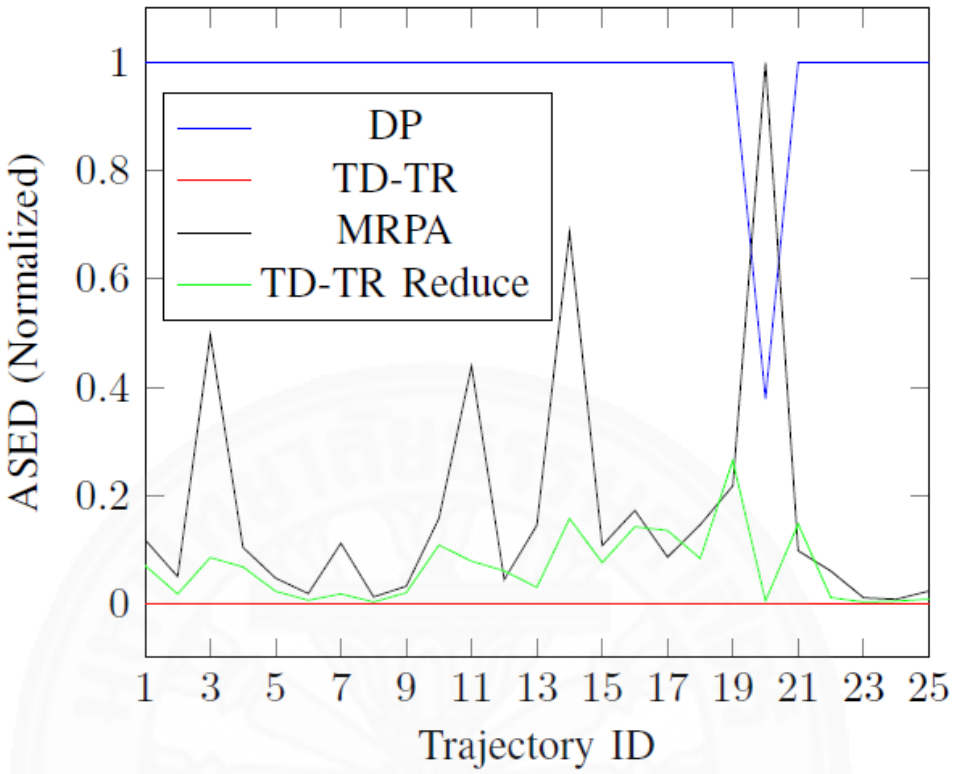

Figure 5.8 Average SED error (ASED)

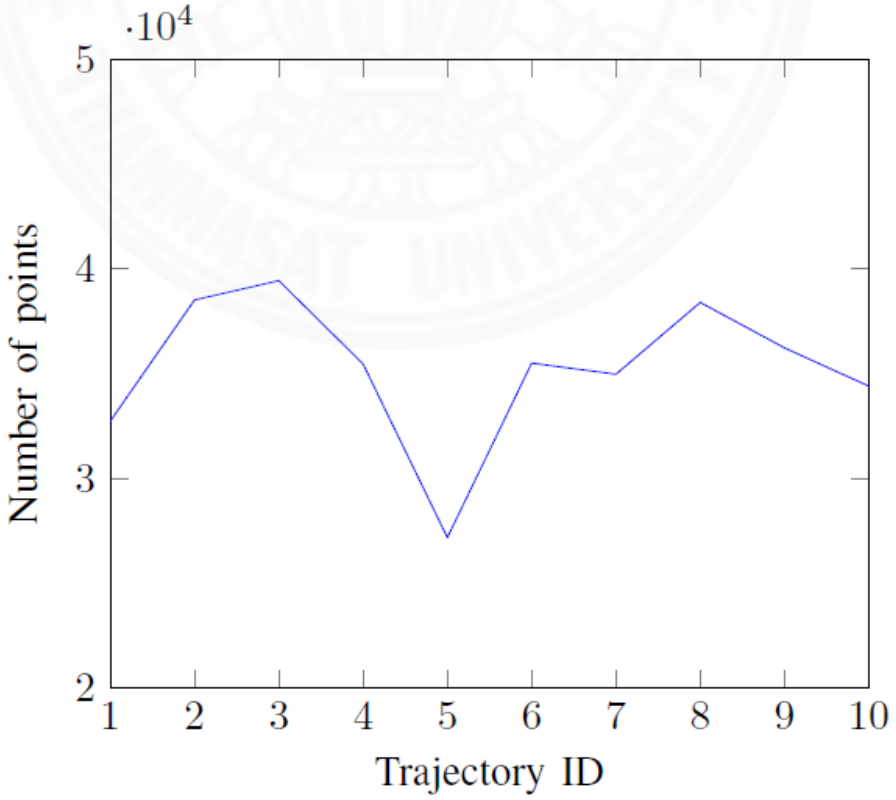

Figure 5.9 Trajectory details

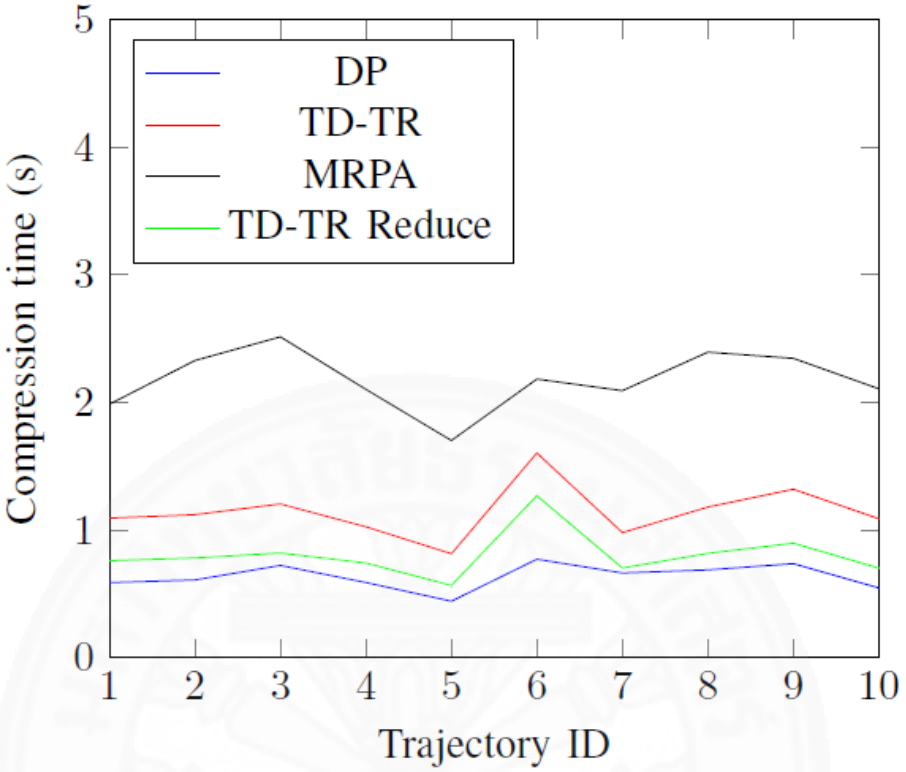

Figure 5.10 Compression time

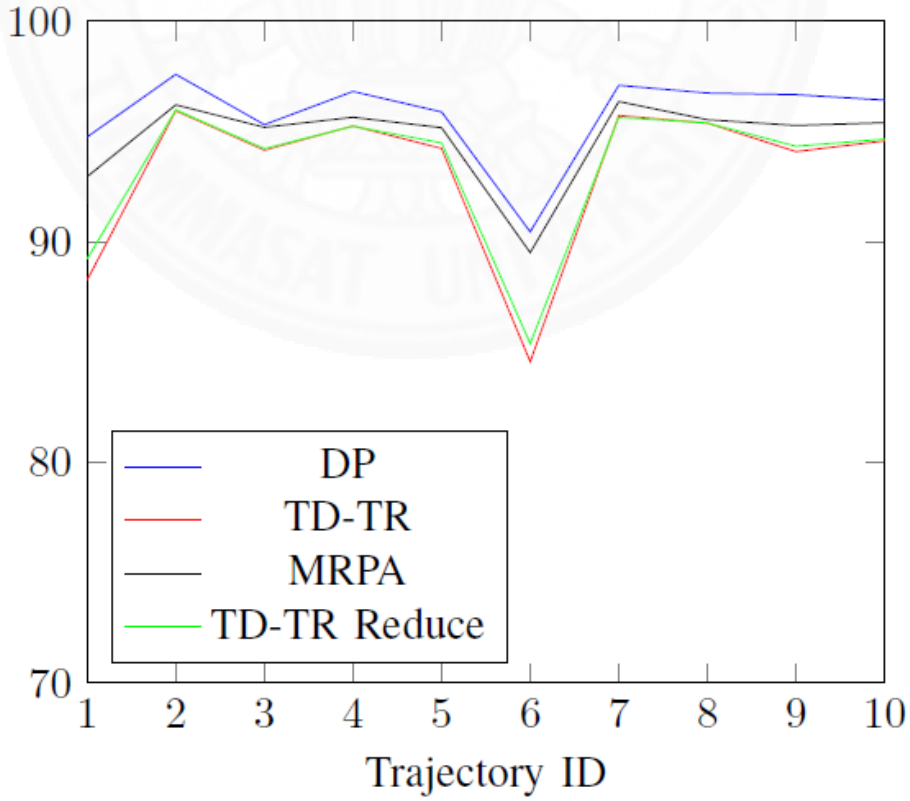

Figure 5.11 Compression ratio

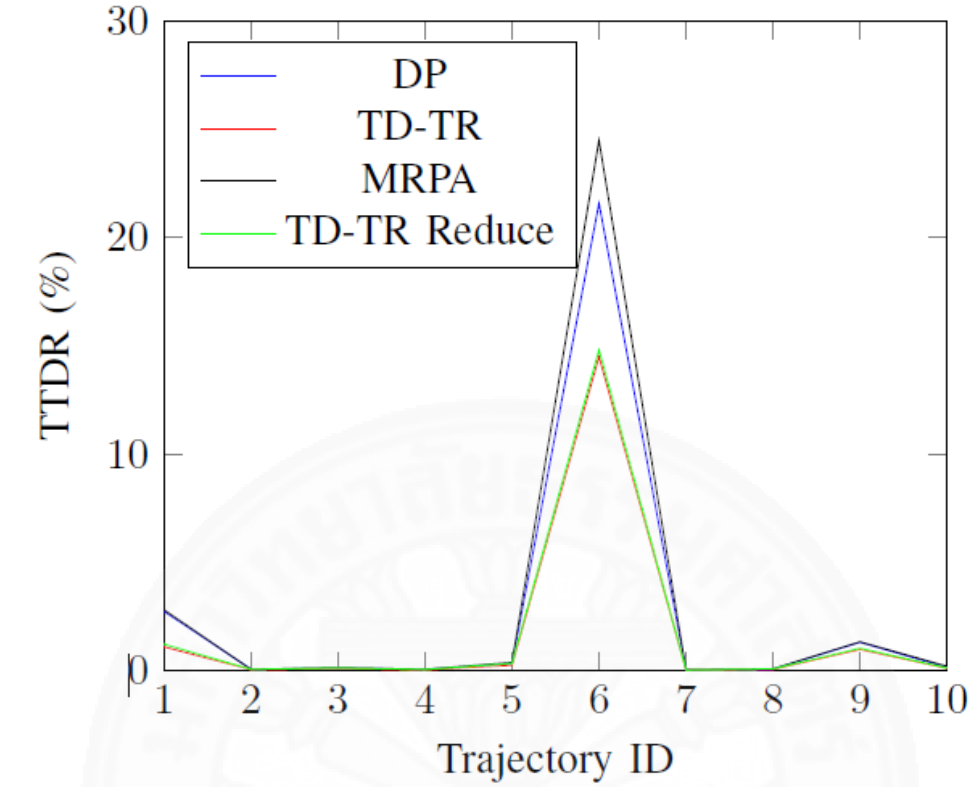

Figure 5.12 Trajectory distance reduction ratio (TDDR)

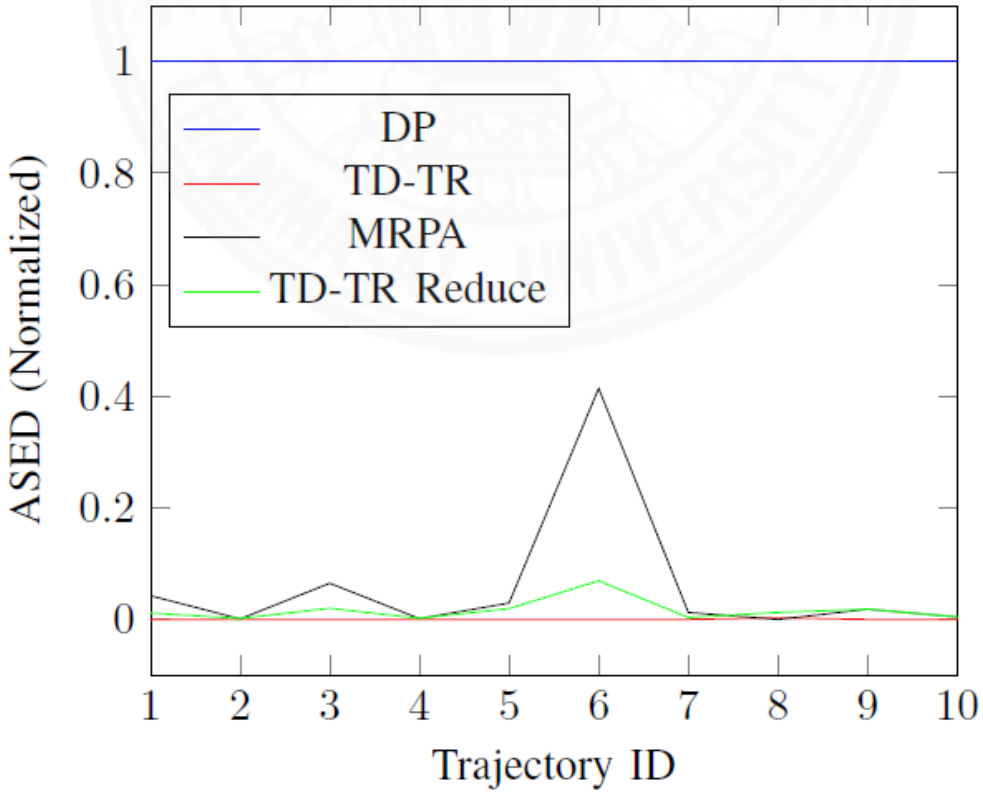

Figure 5.13 Average SED error (ASED)

5.4 Comparison of trajectory simplification algorithm

Our proposed TD-TR Reduce algorithm was compared against three other algorithms (DP, TD-TR, MRPA) in terms of compression ratio, compression time and simplification error. To simulate the result, 25 trajectories are selected from the Geolife dataset and the trajectory details are shown in Figure 5.4. Figures 5.5-5.8 display the simulation results.

As shown in Figure 5.5, on compression time, DP outperforms other algorithms, while the compression time of MRPA is significantly longer than other algorithms. The reason could be that MRPA error metric (LSSD) requires higher computation cost. Furthermore, TD-TR Reduce achieve shorter compression time than both of TD-TR and MRPA. This is because the feature point extraction technique was adopted, which reduces the computational time significantly.

In Figure 5.6, regarding the compression ratio, the plots of DP and MRPA are close to each other, while the plot of TD-TR and TD-TR Reduce are slightly lower. Meanwhile, TD-TR Reduce outperforms the traditional TD-TR algorithm.

Figures 5.7 illustrates that both TD-TR and TD-TR Reduce achieve much lower trajectory distance reduction ratio, while DP obtains the highest trajectory distance reduction ratio among all algorithms.

In Figures 5.8, the ASED error of DP is generally higher than other algorithms and the curves of TD-TR Reduce are slightly higher than Traditional TD-TR. Particularly, TD-TR always achieves the lowest ASED error. Therefore, TD-TR Reduce makes a favorable trade-off between the compression ratio, the trajectory distance reduction ratio and the ASED error while having up to 33% lower compression time on large trajectory compares to the traditional TD-TR algorithm.

To further evaluate the algorithm on large trajectory, 10 large trajectories from the Geolife dataset are selected and the trajectory details are shown in Figure 5.9. The simulation results are then display in Figures 5.10-5.13. The result confirms that our hypothesis is correct, in that our proposed TD-TR Reduce algorithm performs on average 30 percent faster than the traditional TD-TR algorithm on a large trajectory. Therefore, TD-TR Reduce enables a favorable tradeoff between the simplification rate and the simplification error especially on large trajectory.

CHAPTER 6
CONCLUSION

6.1 General overview

This chapter concludes major findings of the study. Contributions to the trajectory simplification community, limitations, and recommendations for further studies are also presented in this chapter.

6.2 Major findings

This study develops the trajectory simplification algorithm utilizing the feature extraction approach to examine performance in real-world data set and suggest some strategies for the trajectory simplification community.

In this paper, we presented a new trajectory simplification algorithm called TD-TR Reduce, which proposes a new method of data reduction based on the extraction of a feature point. Only the important points will be retained by using this data reduction method. The algorithm then performs the traditional TD-TR algorithm on the extracted feature point set. The outstanding advantage of our proposed method is that our proposed algorithm performs exceptionally well around 30% faster on average on large trajectory while still maintaining a low error compare to the traditional algorithm.

6.3 Limitation and recommendations for future studies

There are limitations in this study. As our algorithm is being evaluated on a single data set, some adjustments on experiment setting may need to be made to achieve the desired result. Another limitation is that our algorithm do not perform well on high fluctuation trajectory data.

Future studies should focus on investigating an appropriate dynamic parameter setting (e.g., heading threshold and skip threshold). In addition, an effectiveness of TD-TR Reduce performance on different datasets should be investigated as well.

REFERENCES

Muckell, J., Olsen, P. W., Hwang, J.-H., Lawson, C. T., & Ravi, S. S. (2013). Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica*, 18(3), 435–460. doi: 10.1007/s10707-013-0184-0

Meratnia, N., & By, R. A. D. (2004). Spatiotemporal Compression Techniques for Moving Point Objects. *Advances in Database Technology - EDBT 2004 Lecture Notes in Computer Science*, 765–782. doi: 10.1007/978-3-540-24741-8_44

Zhang, D., Ding, M., Yang, D., Liu, Y., Fan, J., & Shen, H. T. (2018). Trajectory simplification. *Proceedings of the VLDB Endowment, 11(9),* 934–946. doi: 10.14778/3213880.3213885

Douglas, D. H., & Peucker, T. K. (1973). Algorithms For The Reduction Of The Number Of Points Required To Represent A Digitized Line Or Its Caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2), 112–122. doi: 10.3138/fm57-6770-u75u-7727

Chen, M., Xu, M., & Franti, P. (2012). A Fast O(N) Multiresolution Polygonal Approximation Algorithm for GPS Trajectory Simplification. *IEEE Transactions on Image Processing*, 21(5), 2770–2785. doi: 10.1109/tip.2012.2186146

Muckell, J., Hwang, J.-H., Patil, V., Lawson, C. T., Ping, F., & Ravi, S. S. (2011). Squish. *Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications* - COM.Geo 11. doi: 10.1145/1999320.1999333

Zheng, Y., Chen, Y., Li, Q., Xie, X., & Ma, W.-Y. (2010). Understanding transportation modes based on GPS data for web applications. *ACM Transactions on the Web*, 4(1), 1–36. doi: 10.1145/1658373.1658374

Zheng, Y., & Xing Xie, W.-Y. M. (2010) Geolife: a collaborative social networking service among user, location and trajectory, *IEEE Data Engineering Bulletin*, vol. 33, no. 2, pp. 32–40.

APPENDICES

APPENDIX A

SOURCE CODE OF TD-TR ALGORITHM

```
from math import sqrt
import sys
import csv
from math import radians, cos, sin, asin, sqrt
from shapely.geometry import Point
sys.setrecursionlimit(15000)
import time
epsilon = 0.0001
file_in = 'input.csv'

def Calc_SED(start,param,end):
    lat1 = start[0]
    lat2 = param[0]
    lat3 = end[0]
    lon1 = start[1]
    lon2 = param[1]
    lon3 = end[1]
    time1 = start[2]
    time2 = param[2]
    time3 = end[2]
    numerator = int(time2) - int(time1)
    denominator = int(time3) - int(time1)
    if(denominator==0):
      time_ratio = 1
    else:
      time_ratio = numerator / denominator
    lat = float(lat1) + (float(lat3) - float(lat1))*time_ratio
    lon = float(lon1) + (float(lon3) - float(lon1))*time_ratio
    lat_diff = lat - float(lat2)
    lon_diff = lon - float(lon2)
    return sqrt(lat_diff*lat_diff + lon_diff*lon_diff)

def td_tr(points, epsilon):
    dmax = 0.0
    index = 0
    for i in range(1, len(points) - 1):
        d = Calc_SED(points[0],points[i],points[-1])
        if d > dmax:
            index = i
            dmax = d
    if dmax >= epsilon:
        results = td_tr(points[:index+1], epsilon)[:-1] +
td_tr(points[index:], epsilon)
    else:
        results = [points[0], points[-1]]
    return results

start_time = time.time()
all_list = []
with open(file_in) as csvfile:
    readCSV = csv.reader(csvfile, delimiter=',')
    for row in readCSV:
        all_list.append([float(row[0]),float(row[1]),int(row[2])])
```

```
##print(len(all_list))
    result = td_tr(all_list,epsilon)
##    print(result)

file_out = file_in.split(".")[0]+"_TD.csv"

f= open(file_out,"w+")
for i in range(len(result)):
    lat = '%.6f' % result[i][0]
    lon = '%.6f' % result[i][1]
    time2 = result[i][2]
    re_out = str(lat)+','+str(lon)+','+str(time2)
    f.write(re_out)
    if(i!=len(result)-1):
        f.write("\n")
f.close()
print("--- %s seconds ---" % (time.time() - start_time))
```

APPENDIX B

SOURCE CODE OF TD-TR REDUCE ALGORITHM

```
import sys
import csv
import statistics
from math import radians, cos, sin, asin, sqrt
import math
from shapely.geometry import Point
sys.setrecursionlimit(15000)
import time

epsilon = 0.0001
file_in = 'input.csv'

def Calc_SED(start,param,end):
    lat1 = start[0]
    lat2 = param[0]
    lat3 = end[0]
    lon1 = start[1]
    lon2 = param[1]
    lon3 = end[1]
    time1 = start[2]
    time2 = param[2]
    time3 = end[2]
    numerator = int(time2) - int(time1)
    denominator = int(time3) - int(time1)
    if(denominator==0):
      time_ratio = 1
    else:
      time_ratio = numerator / denominator
    lat = float(lat1) + (float(lat3) - float(lat1))*time_ratio
    lon = float(lon1) + (float(lon3) - float(lon1))*time_ratio
    lat_diff = lat - float(lat2)
    lon_diff = lon - float(lon2)
    return sqrt(lat_diff*lat_diff + lon_diff*lon_diff)

def td_tr(points, epsilon):
    dmax = 0.0
    index = 0
    for i in range(1, len(points) - 1):
        d = Calc_SED(points[0],points[i],points[-1])
        if d > dmax:
            index = i
            dmax = d
    if dmax >= epsilon:
        results = td_tr(points[:index+1], epsilon)[:-1] +
td_tr(points[index:], epsilon)
    else:
        results = [points[0], points[-1]]
    return results


def Calc_speed(lat1, lon1, time1, lat2, lon2, time2):
```

```
        R = 6371
        dLat = radians(float(lat2) - float(lat1))
        dLon = radians(float(lon2) - float(lon1))
        lat1 = radians(float(lat1))
        lat2 = radians(float(lat2))
        timediff = int(time2)-int(time1)

        a = sin(dLat/2)**2 + cos(lat1)*cos(lat2)*sin(dLon/2)**2
        c = 2*asin(sqrt(a))
        if(timediff==0):
            result = 0
        else:
            result = (R * c * 1000)/timediff*3.6

        return result #meter

def Calc_bearing(lat1, lon1, lat2, lon2):
        lat1 = math.radians(float(lat1))
        lat2 = math.radians(float(lat2))
        lon1 = math.radians(float(lon1))
        lon2 = math.radians(float(lon2))
        y= sin(lon2-lon1)*cos(lat2)
        x= cos(lat1)*sin(lat2)-sin(lat1)*cos(lat2)*cos(lon2-lon1)
        bearing = math.atan2(y,x)
        return (bearing + math.radians(360)) % math.radians(360)

def Calc_bearing_diff(b1,b2):
        if(abs(b1-b2)>180):
              return 360-abs(b1-b2)
        else:
              return abs(b1-b2)
        return (bearing + math.radians(360)) % math.radians(360)


start_time = time.time()
all_list = []
speed_list = []
bearing_list = []
list1=[] #p1
fin_list = []
skip = 0
with open(file_in) as csvfile:
    readCSV = csv.reader(csvfile, delimiter=',')
    for row in readCSV:
        if(len(list1)!=0):
            speed =
Calc_speed(list1[0],list1[1],list1[2],row[0],row[1],row[2])
            bearing = Calc_bearing(list1[0],list1[1],row[0],row[1])
            speed_list.append(speed)
            bearing_list.append(math.degrees(bearing))

all_list.append([float(row[0]),float(row[1]),int(row[2])])
            list1=row
        else:
            list1=row

all_list.append([float(row[0]),float(row[1]),int(row[2])])
```

```
mean = statistics.mean(speed_list)
SD_speed = statistics.stdev(speed_list)
fin_list.append(all_list[0])
for i,j in zip(range(0,len(bearing_list)-1),
range(1,len(bearing_list))):
    if(abs(speed_list[i]-speed_list[j])>=SD_speed or
Calc_bearing_diff(bearing_list[i],bearing_list[j])>=60):
        skip=0
        fin_list.append(all_list[i+1])
    else:
        skip+=1
    if(skip==5):
        fin_list.append(all_list[i+1])
        skip=0
fin_list.append(all_list[-1])
result = td_tr(fin_list,epsilon)

file_out = file_in.split(".")[0]+"_TD_RED.csv"

f= open(file_out,"w+")
for i in range(len(result)):
    lat = '%.6f' % result[i][0]
    lon = '%.6f' % result[i][1]
    time2 = result[i][2]
    re_out = str(lat)+','+str(lon)+','+str(time2)
    f.write(re_out)
    if(i!=len(result)-1):
        f.write("\n")
f.close()
print("--- %s seconds ---" % (time.time() - start_time))
```

APPENDIX C

SOURCE CODE OF DP ALGORITHM

```
from math import sqrt
import sys
import csv
from math import radians, cos, sin, asin, sqrt
from shapely.geometry import Point
sys.setrecursionlimit(15000)
import time

epsilon = 0.0001
file_in = 'input.csv'
def distance(a, b):
    return  sqrt((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2)

def point_line_distance(point, start, end):
    if (start == end):
        return distance(point, start)
    else:
        n = abs(
            (end[0] - start[0]) * (start[1] - point[1]) - (start[0] -
point[0]) * (end[1] - start[1])
        )
        d = sqrt(
            (end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2
        )
        return n / d

def rdp(points, epsilon):
    dmax = 0.0
    index = 0
    for i in range(1, len(points) - 1):
        d = point_line_distance(points[i], points[0], points[-1])
        if d > dmax:
            index = i
            dmax = d
    if dmax > epsilon:
        results = rdp(points[:index+1], epsilon)[:-1] +
rdp(points[index:], epsilon)
    else:
        results = [points[0], points[-1]]
    return results

start_time = time.time()
all_list = []
with open(file_in) as csvfile:
    readCSV = csv.reader(csvfile, delimiter=',')
    for row in readCSV:
        all_list.append([float(row[0]),float(row[1]),int(row[2])])
##print(len(all_list))
result = rdp(all_list,epsilon)
```

```
file_out = file_in.split(".")[0]+"_DP.csv"

f= open(file_out,"w+")
for i in range(len(result)):
    lat = '%.6f' % result[i][0]
    lon = '%.6f' % result[i][1]
    time2 = result[i][2]
    re_out = str(lat)+','+str(lon)+','+str(time2)
    f.write(re_out)
    if(i!=len(result)-1):
        f.write("\n")
f.close()
print("--- %s seconds ---" % (time.time() - start_time))
```

APPENDIX D

SOURCE CODE OF ERROR CALCULATION

```
import sys
import csv
from math import radians, cos, sin, asin, sqrt

def Calc_PED(lat1, lon1, lat2, lon2, lat3, lon3):
    if((lat1==lat2 and lon1==lon2) or (lat2==lat3 and lon2==lon3)):
        return 0.0
    else:
        A = float(lon3) - float(lon1)
        B = float(lat1) - float(lat3)
        C = float(lat3)*float(lon1) - float(lat1)*float(lon3)
        if(A==0 and B==0):
            return 0;
        shortDist = abs((A * float(lat2) + B * float(lon2) + C) /
sqrt(A * A + B * B));
        return shortDist

def Calc_SED(lat1, lon1, time1, lat2, lon2, time2, lat3, lon3,
time3):
  numerator = int(time2) - int(time1)
  denominator = int(time3) - int(time1)
  if(denominator==0):
      time_ratio = 1
  else:
      time_ratio = numerator / denominator
  lat = float(lat1) + (float(lat3) - float(lat1))*time_ratio
  lon = float(lon1) + (float(lon3) - float(lon1))*time_ratio
  lat_diff = lat - float(lat2)
  lon_diff = lon - float(lon2)
  return sqrt(lat_diff*lat_diff + lon_diff*lon_diff)

def Calc_speed(lat1, lon1, time1, lat2, lon2, time2):

      R = 6371
      dLat = radians(float(lat2) - float(lat1))
      dLon = radians(float(lon2) - float(lon1))
      lat1 = radians(float(lat1))
      lat2 = radians(float(lat2))
      timediff = int(time2)-int(time1)

      a = sin(dLat/2)**2 + cos(lat1)*cos(lat2)*sin(dLon/2)**2
      c = 2*asin(sqrt(a))

      return (R * c * 1000)/timediff*3.6 #meter

def Calc_dis(lat1, lon1, lat2, lon2):

      R = 6371

      dLat = radians(lat2 - lat1)
      dLon = radians(lon2 - lon1)
      lat1 = radians(lat1)
```

```
      lat2 = radians(lat2)

      a = sin(dLat/2)**2 + cos(lat1)*cos(lat2)*sin(dLon/2)**2
      c = 2*asin(sqrt(a))

      return R * c * 1000

Traj = 'original.csv'
STraj = 'simplified.csv'
list1=[]
list2=[]
Traj_list = []
Straj_list = []
total_ped = 0
total_sed = 0
total_dist1 = 0
total_dist2 = 0
num_line1 = 0
done = True
safe_line = 1
with open(STraj) as before:
  csv_before = csv.reader(before, delimiter=',')
  for row in csv_before:
    list2=list1
    list1=row
    num_line1+=1
    Straj_list.append(row)
    if(len(list1)!=0 and len(list2)!=0):
      with open(Traj) as after:
        csv_after = csv.reader(after, delimiter=',')
        for row2 in csv_after:
          if(list1!=row2):
            if(csv_after.line_num>safe_line):
              total_sed = total_sed + Calc_SED(list2[0], list2[1],
list2[2], row2[0], row2[1], row2[2], list1[0], list1[1], list1[2])
            else:
              safe_line=csv_after.line_num
              break

with open(Traj) as after:
  csv_after = csv.reader(after, delimiter=',')
  for row in csv_after:
    Traj_list.append(row)

for i in range(len(Traj_list)):
    for j in range(len(Traj_list)):
        if(j-i==1):
            total_dist1 = total_dist1 +
Calc_dis(float(Traj_list[i][0]),float(Traj_list[i][1]),float(Traj_lis
t[j][0]),float(Traj_list[j][1]))

for i in range(len(Straj_list)):
    for j in range(len(Straj_list)):
        if(j-i==1):
            total_dist2 = total_dist2 +
Calc_dis(float(Straj_list[i][0]),float(Straj_list[i][1]),float(Straj_
list[j][0]),float(Straj_list[j][1]))
```

```
print("File Name",STraj)
print("ASED:",total_sed/(num_line1-1))
print("Distance Reduction Ratio:",1-(total_dist2/total_dist1))
```

BIOGRAPHY

Name	Mr. Kanasuan Hansuddhisuntorn
Date of Birth	March 29, 1995
Education	2013: Bachelor of Engineering (Computer Engineering) Sirindhorn International Institute of Technology Thammasat University 2017: Master of Science (Engineering and Technology) Sirindhorn International Institute of Technology Thammasat University
Publications	

Hansuddhisuntorn K. & Horanont T., (2019). Improvement of TD-TR Algorithm for Simplifying GPS Trajectory Data. *The First International Conference on Smart Technology & Urban Development (STUD 2019)*