Trajectory Simplification Method for Location-Based Social Networking Services

Yukun Chen[1], Kai Jiang[3], Yu Zheng[2], Chunping Li[1], Nenghai Yu[3]

[1]School of Software, Tsinghua University, Beijing 100084, China

[2]Microsoft Research Asia, 4F, Sigma Building, NO. 49 Zhichun Road, Haidian District, Beijing 100190, China

[3]MOE-MS Key Lab of MCC, University of Science and Technology of China, Hefei 230027, China.

[1]chenyukun03@hotmail.com, [3]jkustc@gmail.com, [2]yuzheng@microsoft.com, [1]cli@tsinghua.edu.cn, [3]ynhustc@gmail.com

ABSTRACT

The increasing availabilities of GPS-enabled devices have given rise to the location-based social networking services (LBSN), in which users can record their travel experiences with GPS trajectories and share these trajectories among each other on Web communities. Usually, GPS-enabled devices record far denser points than necessary in the scenarios of GPS-trajectory-sharing. Meanwhile, these redundant points will decrease the performance of LBSN systems and even cause the Web browser crashed. Existing line simplification algorithms only focus on maintaining the shape information of a GPS trajectory while ignoring the corresponding semantic meanings a trajectory implies. In the LBSN, people want to obtain reference knowledge from other users' travel routes and try to follow a specific travel route that interests them. Therefore, the places where a user stayed, took photos, or changed moving direction greatly, etc, would be more significant than other points in presenting semantic meanings of a trajectory. In this paper, we propose a trajectory simplification algorithm (TS), which considers both the shape skeleton and the semantic meanings of a GPS trajectory. The heading change degree of a GPS point and the distance between this point and its adjacent neighbors are used to weight the importance of the point. We evaluated our approach using a new metric called normalized perpendicular distance. As a result, our method outperforms the DP (Douglas-Peuker) algorithm, which is regarded as the best one for line simplification so far.

Categories and Subject Descriptors

G.1.2 [Approximation]: Special function approximations – *line simplification*.

General Terms

Algorithms, Measurement, Experimentation, Performance.

Keywords

Trajectory simplification, GPS trajectory, Location-based social networking service.

1. INTRODUCTION

Location-acquisition technologies, such as GPS and GSM network, enable users to obtain their locations and record travel experiences with some trajectories. Social networks based on user-generated GPS data [1][2][3][4][12][14]. have been surging on the Web and showing great potential to change people's lives. In such applications, users first record their trajectory using GPS-enabled devices, then upload, visualize, browse and share those trajectories in an Internet community. From other's trajectories, people are more likely to find the travel routes that interest them and acquire reference knowledge facilitating their travel.

Usually, GPS-enabled devices record far more data than necessary. For instance, if a GPS device is configured to record a GPS point every 2 seconds, a one-day hiking route could be comprised of 10,000 points. Such dense representation is not necessary for trajectory visualization and sharing. Moreover, these redundant GPS points will decrease the performance of the LBSN systems in the following aspects. First, this will waste a large amount of storage**.** Second, these redundant trajectories will cause a heavy load for network transfer. As most trajectory sharing applications are based on Internet, the more GPS points a trajectory contains the longer uploading/downloading time a user need to spend. Third, this dense representation will bring a heavy burden for web browser for rendering trajectories. Sometimes, Web browsers would be out of memory and hence get crashed. To ensure the usability, Google Map and Bing Map only allow a user to show limited points per track.

Line simplification methods [5][7][8], such as DP algorithm [5], which are widely used in computer aided design (CAD) area, can be employed to simplify a GPS trajectory. However, those algorithms focus only on maintaining the shape of a trajectory while overlooking the semantic meaning of the trajectory.

For example, as shown in Figure 1, a trajectory consists of two parts: a long driving segment on the highway and a short winding walking segment in an interesting travel spot. For the sake of performance, we can only maintain limited number of points when displaying this trajectory on a web map. According to DP algorithm's metrics called perpendicular distance, more GPS points will be maintained in the driving segment while few points would be left for the walk segment (refer to Figure 2). As the distance between consecutive points from a driving segment would be larger than that of walking segments, losing a driving point is more likely to cause a larger perpendicular distance; hence losing more information of the skeleton. The simplified result is acceptable in CAD area.

However, people intending to follow the trajectory do not only need the structure information of the whole trajectory but also care about the walk segment containing rich semantic meanings, e.g., the places where the user stayed, took photos or watching something attractive, etc. If this trajectory is simplified by DP algorithm (refer to Figure 2), people could know little about how the trajectory-creator traveled on the island (see the original track shown in Figure 1 a)) as well as his/her travel route along the lake (see Figure 1 b)). Without these details, people will miss lots of interesting attractions on the trip and can obtain little reference

knowledge from the creator's travel experiences. On the contrary, people still can clearly understand the creator's driving direction even if the simplified driving segment is not perfectly aligned with the road networks (refer to Figure 3 for the trajectory simplified by our method). In short, a relatively big deviation like 100 meters on a driving segment would not reduce the semantic meanings a trajectory implies, while a small deviation from the walk segment would loss the significant information about a trip (e.g., sometimes 20 meters derivation will bring people to two different routes when walking).

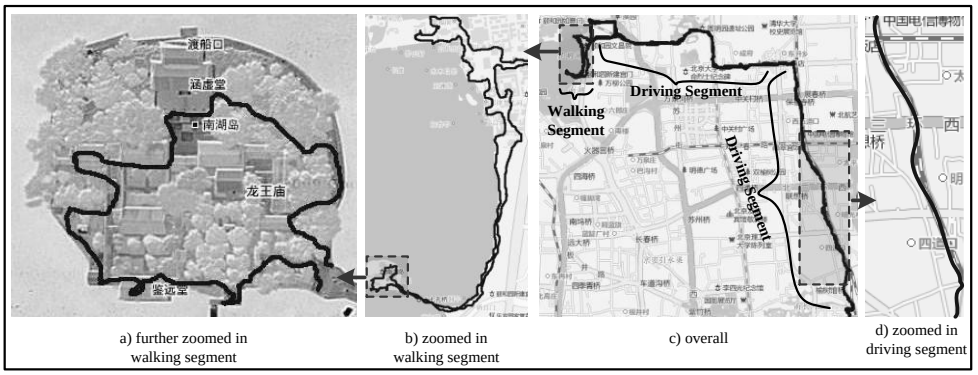

Figure 1: A GPS trajectory containing a walking segment and a driving segment.

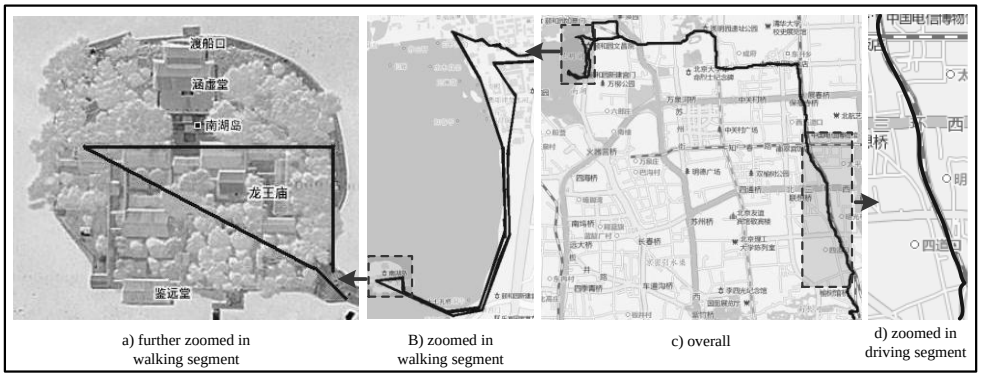

Figure 2: The GPS trajectory after being simplified by the DP algorithm

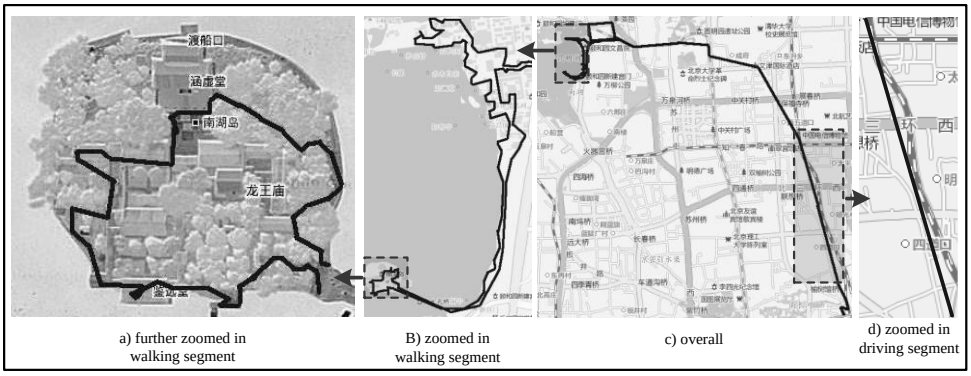

Figure 3: The GPS trajectory after being simplified by the TS algorithm.

In this paper, we propose a GPS trajectory simplification (TS) algorithm for Location-Based Social Networking Services. The TS aims to simplify a trajectory with a large number of points to limited ones and maintain the information of both skeleton and semantics of a trajectory. The heading change degree of a GPS point and the distance between this point and its most adjacent neighbors (called neighbor distance) are used to weight the importance of the point. At the same time, a novel measurement, referred to as normalized perpendicular distance, is proposed to evaluate the effectiveness of line simplification in the scenarios of GPS-trajectory-sharing.

This weighting strategy is motivated by the following two observations. First, typically when people are attracted by something, take photos or stay somewhere for a while, they are more likely to change their moving directions (called heading change in this paper) than walking. Meanwhile, a relatively big turn in a driving segment would also be important in maintaining the key directions of a driving way. In other words, the bigger the

heading change degree is the more important semantic meanings this point could carry and the more skeleton information this point could maintain. Second, points crowdedly distributed in a short distance are redundant to represent the skeleton and semantic meanings of a trajectory. Consequently, the point with a relatively big neighbor distance is assigned with a big weight.

The TS algorithm is comprised of four steps: segmentation, point distribution, point weighting and point selection. At first, we partition a GPS trajectory into walking and non-walking segments using a method we proposed in paper [12][14]. Then, the headcounts of points will be assigned to each segment in terms of the product of the average heading change and the distance of each segment. In each segment, the points with a relatively large production of heading change and neighbor distance will be assigned a big weight. Finally, the top points with large weights are remained to represent the simplified trajectory. The contributions of this paper lie in two aspects:

- We devise a trajectory Simplification method (TS) maintaining both the skeleton and semantic meanings of a trajectory for location-based social networking services.
- We propose a normalized perpendicular distance to measure the effectiveness of line simplification algorithms in the scenario of GPS-trajectory-sharing.
- We evaluate our method with large-scale GPS trajectories generated by people traveling in the real world. As a result, our algorithm outperforms the DP algorithm with a 59% improvement.

The remainder of this paper is organized as follows. First some related work about line simplification is reported and compared with our approach in Section 2. After that we propose our method (TS) for line simplification in Section 3. Next, using a normalized perpendicular distance, we evaluated our algorithm and compared it with the DP algorithm in Section 4. At last in Section 5, we conclude the paper and point out the future work.

2. RELATED WORK

Traditional line simplification algorithms can be categorized into two classes: 1) Local process method; 2) Global process method.

Local process methods: Local process methods simplify trajectory by checking only neighboring points. A simple fixed *nth* method [12] deletes all but every *nth* point along the line where n is a fixed integer based on the desired degree of simplification. Tobler [8] proposed a method to filter points using a pre-set distance threshold. If the current point is within the pre-set distance threshold from previous neighboring point, it's kept, otherwise removed. Lang (1969) [7] described another algorithm searching beyond the immediate neighbors and was reported by Douglas and Peuker (1973) 'as producing an acceptable result'. The objective of the procedure was to delete points if they were found to lie within a tolerance distance of a straight line segment being tested to represent a line. From one representative point it constructs straight lines to subsequent points until one point between the representative point and the sub-point is further away from the line linking the two than a pre-set tolerance value. As soon as this condition is satisfied, the point before the sub-point becomes a new representative point and the procedure is repeated. This method gives acceptable results in the case of smooth curves but yields bad results in the case of sharp angles. Another interesting idea is to map all the points on to some screen coordinates, many points falling into one screen solution can be eliminated. W. R. Tobler described this method in 1966 [8]. Locally processing is very fast in execute time, however it's very difficult to find the optimized result.

Our approach adopts the local process idea when weighting each point inside a segment. It considers some neighbor points around the current point and sequentially processes each point. However, unlike just using a single feature [7][8], we define two new features to identify the importance of a point. They are combined to maintain both the semantics and skeleton of the trajectory.

Global process methods: global process methods consider the line as a whole while processing. The DP algorithm [5] developed by Douglas and Peuker, in 1973, is the only global process algorithm so far. As shown in Figure 4, at the beginning a tolerance distance (td) is per-defined. The first point a is selected as the anchor point and the last point b is selected as the float point. The maximum perpendicular distance, say fc(labeled in the curve of Figure 4), from the digitised points to the line ab is compared with td. If $fc > td$, then c becomes the new float point. Then, the maximum perpendicular distance from points to ac, say ed, is compared with td. If $ed > td$, then d becomes the new floating point, c is saved in the first position on a stack, or else d is selected as the new anchor point and all points between a and d are discarded. In this fashion the program processes the entire line, backing up onto the stack as necessary, until all intermediary points are within the corridor, or there are no intermediary points.

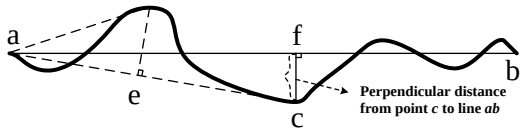

Figure 4 Example of the DP algorithm

The last float point now becomes the anchor and the first co-ordinate pair is taken from the stack to become the new float point. The process described in the previous paragraph is now repeated. This recursive method continues until the last co-ordinate pair is taken from the stack. A stack may be considered as a list that is accessed from only one end. Items are taken from a stack in reverse to the order in which they are placed on it.

The DP algorithm is widely used in cartography related software like AUTOCAD. It produces the best results in terms of both vector displacement and area displacement [10]. "Many cartographers consider it to be the most accurate simplification algorithm available, while others think that it is too slow and costly in terms of computer processing time" [11]. Anyway, it's the most famous line simplification algorithm till now.

Our method differs from the DP in two aspects. First, TS combines the strategy of local process with global process. In TS, a trajectory is first partitioned into several segments, which will be weighted in a global view. Then, points from one segment will be ranked locally in terms of the feature we defined in Section 3. Second, TS considers both the skeleton information and semantic meanings of a trajectory when performing simplification. Therefore, this algorithm is more proper beyond the DP methods for location-based social networking services, where people intend to obtain reference knowledge from and follow a user's travel route. Overall, TS outperforms DP algorithms with a better efficiency.

3. PRELIMINARY

In this section, we elaborate our TS algorithm used in trajectory simplification. First we clarify some terms used in this paper. Afterwards, the problem statement is presented. At last we give an overview of our approach for understanding.

3.1 Definitions

Definition 1. GPS log and GPS trajectory: As shown in Figure 5, a GPS log P is a sequence of GPS points. $P = \{p_1, p_2, \ldots p_m\}$, each GPS point *p* contains latitude, longitude and timestamp. On a two dimensional plane, we can sequentially connect these GPS points into a GPS trajectory.

Definition 2. Segment: A segment represents a part of trajectory where a user travels on foot or by other transportation modes like driving. We name the former kind of segment *Walk* segment, and call the rest *non-Walk* segment.

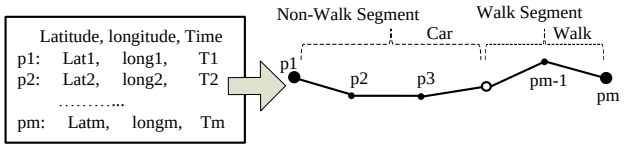

Figure 5: GPS logs, GPS trajectory and segment

Definition 3. Heading Direction. Heading direction h denotes the direction of a moving object at a specific point location using north direction as a basis, where $0° \leq h < 360°$.

***Definition 4. Neighbor Heading Change*:** Neighbor heading change θ of a point p_i is the difference between its heading direction $p_i.h$ and that of its nearest previous point p_{i-1}, i.e., $\theta = p_i.h - p_{i-1}.h$, where $-180^\circ < \theta < 180^\circ$.

***Definition 5. Accumulated Heading Change*:** The accumulated heading change β of the *i*th point (p_i) of a trajectory is defined as:

$$p_i.\beta = \sum_{k=i-\tau}^{i+\tau} p_k.\theta, \quad (1)$$

where $p_k.\theta$ is the neighbor heading change of p_k, and τ is a pre-set integer threshold denoting half of number of neighbor points.

***Definition 6. Heading change*:** The heading change γ of a point p is the sum of the absolute value of the neighbor heading change and that of the accumulated heading change,

$$p.\gamma = |p.\theta| + |p.\beta|. \quad (2)$$

Definition 7. Neighbor Distance: The neighbor distance d of a point p_i is the sum of Euclid distance between p and its two nearest neighbor points.

$$p_i.d = Distance(p_{i-1}, p_i) + Distance(p_i, p_{i+1}), \quad (3)$$

where $Distance(p_{i-1}, p_i)$ is a function used to calculate the Euclid distance between two points.

***Example* 1.** As shown in Figure 6 a), a GPS trajectory is comprised of five points (p_1, p_2, p_3, p_4, p_5). $p_2.h$ and $p_3.h$ respectively represent the heading direction of p_2 and p_3, and $p_3.\theta = p_3.h - p_2.h$ is the neighbor heading change of p_3, where $-180^\circ < p_3.\theta < 180^\circ$. $p_2.d = d_1 + d_2$ is the distance change of p_2. As sharp turning points in a trajectory may have a very big neighbor heading, $p_3$could be distinguished from others.

As demonstrated in Figure 6 b). The neighbor heading changes of point p_1, p_2, p_3, p_4, p_5 are $p_1.\theta, p_2.\theta, p_3.\theta, p_4.\theta, p_5.\theta$ respectively. If we set τ as 2, the accumulated heading change of p_3 is $p_3.\beta = |p_1.\theta + p_2.\theta + p_3.\theta + p_4.\theta + p_5.\theta|$.

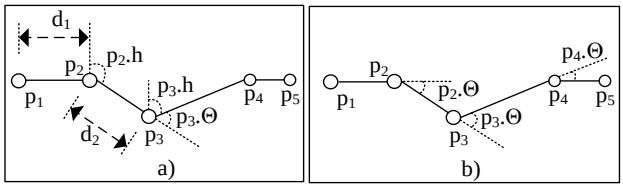

Figure 6: An example of definitions

3.2 Problem Statement

Given a trajectory $Traj$ of n points, our approach aims to select the most representative m points, where $m < n$, from $Traj$ to formulate a new trajectory $Traj'$, which could differ from $Traj$ as small as possible in both skeleton information and semantic meanings. We call $r = m/n$, simplification rate ($0 < r < 1$).

***Example* 2.** As shown in Figure 7, three simplified results based on different *r* are displayed below raw trajectory in a. As shown in Figure 5 b), c) and d), *m* equals to 40, 20 and 5 respectively.

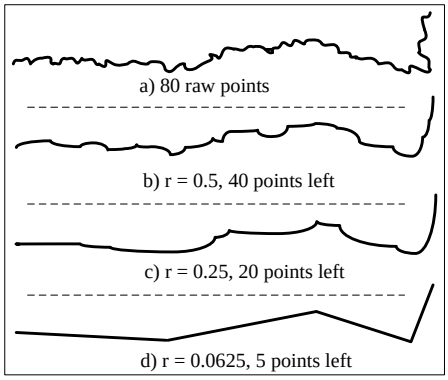

Figure 7: Trajectory simplified by different rate

The bigger *r* is, the more detail the result will loss. *r* should be decided before simplification according to needs.

3.3 The Algorithm Overview

As shown in Figure 8, the architecture of algorithm is comprised of the following four parts: Segmentation, Segment Ranking, Point Weighting and Point Selection. First, Segmentation in line 1 detects all the walk and non-walk segments using algorithm proposed in previous work [12][14], which is quite accurate even in bad traffic situation and variable weather. Second, DistributePoints in line 2 assigns each segment with different headcounts of points in terms of the product of the average heading change and the distance of each segment. The distance of a segment is used to maintain the skeleton while the average heading change can be utilized to maintain the semantics meaning. Third by WeightPoints in line 3, the points with a relatively large production of heading change and neighbor distance will be assigned a big weight. By this means the contribution to the maintenance of skeleton and semantic meaning can be balanced. Finally, by SelectPoints in line 4, the top points with large weights are remained to represent the simplified trajectory.

Algorithm TS(*Traj, m*)

Input: An original Trajectory *Traj* and the number of points *m* of the simplified trajectory.

Output: A simplified trajectory *Traj'* with m points.

1. *Traj'*=∅;
2. *Seg[]* = Segmentation(*Traj*); //Partition *Traj* into segments
3. DistributePoints(*Seg[], m*); //rank segments and assign headcount
4. **Foreach** segment *s* in *Seg[]*
5. WeightPoints(*s*); //give each point a weight
6. *s'*= SelectPoints(*s*); //select points to compose a new segment
7. *Traj' = Traj* ∪ *s'*; //integrate these segments into a new trajectory
8. **Return** *Traj';*

Figure 8: The flowchart of the TS Algorithm

4. TRAJECTORY SIMPLIFICATION

4.1 Segmentation

Using the change point-based segmentation method proposed in paper [12][14], we partition the trajectory by walk segments and non-walk segments. This approach is derived from the following commonsense of real world. 1) Typically, people must stop and then go when changing their transportation modes. 2) Walk should be a transition between different transportation modes. In other words, the start point and end point of a *Walk Segment* could be a change point in very high probability. As demonstrated in Figure 9, we briefly demonstrate the four steps of the segmentation algorithm. For more information please refer to paper [12][14].

Step 1: Using a loose upper bound of velocity (V_t) and that of acceleration (a_t) to distinguish all possible *Walk Points* from *non-Walk Points*.

Step 2: If the length of a segment composed by consecutive *Walk Points* or *non-Walk Points* is less than a threshold, merge this one into its backward segment.

***Step 3*:** If the length of a segment exceeds a certain threshold, the segment is regarded as a *Certain Segment.* Otherwise it is deemed as an *Uncertain Segment*. If the number of consecutive *Uncertain Segment* exceeds a certain threshold, these *Uncertain Segments* will be merged into one *non-Walk Segment.*

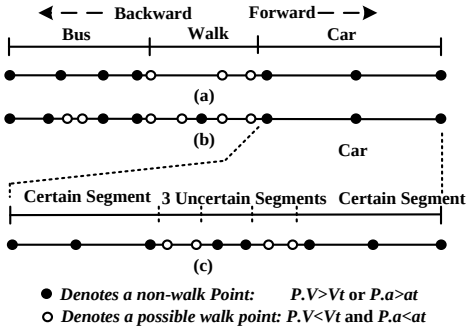

Figure 9: An example of segmentation

The totally cost lies in step 1, which is *O(n).*

4.2 Points Distribution

This step tries to balance the overall skeleton and semantic meaning of a trajectory by combining segment's average heading change and its distance. Usually walking segments have large average heading change while driving segments are longer. Therefore walking segment contains more semantic meanings while driving segment contributes more to skeleton. So both heading change and distance are considered to weight a segment.

After previous segmentation step, the trajectory is partitioned into some walk or non-walk segments. Based on that, we weight each segment by its distance and average heading change, which is the mean value of all the absolute values of neighbor heading changes in the segment. Suppose a trajectory $Traj = (S_1 S_2 \dots S_k \dots S_q)$, S is a segment of *Traj*. S contains *Traj*'s points from p_i to p_j, $S = (p_i p_{i+1} \dots p_{j-1} p_j)$. d is the total distance of S and α is the average heading change of *S:*

$$S.d = \sum_{k=i+1}^{j} Distance(p_k, p_{k-1}) \quad (4)$$

$$S.\alpha = (\sum_{k=i}^{j} |p_k.\theta|) \ / \ (j - i + 1) \quad (5)$$

where $p_k.\theta$ is the neighbor heading change based on definition 4.

The bigger $S.d$ is the more skeleton information this segment could maintain, and the bigger $S.\alpha$ is the more semantic meanings this segment could carry and the more skeleton information this point could maintain, both will strengthen the weight of current segment. We use the product of $S.d$ and $S.\alpha$ to weight a segment:

$$S.w = S.d * S.\alpha, \quad (6)$$

where $S.w$ is the weight of segment S. In order to assign point headcounts conveniently, we normalize $S.w$ by

$$S.w = S.w \ / \ (\sum_{k=1}^{q} S_k.w), \quad (7)$$

where S_k is one of the segment of *Traj* who has totally q segments. After weights of all segments are normalized, each segment is assigned with the proper point headcounts by its normalized weight:

$$S.hc = m * S.w, \quad (8)$$

where $S.hc$ is the points headcounts of S and m is the target number of points in the simplified trajectory. We assign points headcounts in this way because by combining $S.d$ and $S.\alpha$, we can maintain the overall skeleton. Meantime the case is avoided that some straight but long segment with many redundant points is given too many headcounts.

As shown in Figure 10, Line 3 and 4 calculate $S.d$ and S.α based on (4) and (5). The cost of DistributePoints lies in line 3 and line 4, where all points in segment are checked to get the distance and average heading change. Therefore the cost of this step is *O(n).*

Algorithm DistributePoints(Seg[], m)

Input: Segments in a trajectory and the number of points *m* of the simplified trajectory.

Output: No direct output, but distribute points headcount to segments

1. totalWeight = 0;
2. **Foreach** *S in Seg[]*
3. $S.d = \text{GetDistance}(S)$; //distance of segment
4. $S.\alpha = \text{GetHeadChange}(S)$; //average heading change
5. $S.w = S.d * S.\alpha$; //weight segment
6. totalWeight = $\sum_{i=1}^{q} S[i].w$;
7. **Foreach** *S in Seg[]*
8. $S.w = S.w$ / totalWeight ; //normalize weight
9. $S.hc = m * S.w$; //assign headcount

Figure 10: The procedure of points distribution

4.3 Point Weighting

In this sub-section, we aim to give all points inside each segment a weight, for later step to select the best points in each segment. Point is weighted by two observations:

- Typically, when people are attracted by something, take photos or stay somewhere for a while, they are more likely to change their moving directions than normal walking. Meanwhile, a relatively big turn in a driving segment would also be important in maintaining the key directions of a driving way. In other words, the bigger the heading change degree is the more important semantic meanings this point could carry and the more skeleton information this point could maintain.
- Many points will gather at the places where people stay. For the sake of maintaining the skeleton, those points are less important than points having relative large distance with adjacent points. In short, the bigger the neighbor distance is the more skeleton information this point could maintain.

Based on the two observations, we give each point a weight indicating how important it is. Suppose point p with heading change $p.\gamma$ and neighbor distance $p.d$, we formulate its weight:

$$p.w = p.d * p.\gamma \quad (9)$$

$p.d$ and $p.\gamma$ for all points can be calculated by scanning from p_1 to p_n once, so the cost of point weighting is *O(n).*

4.4 Point Selection

Based on Equation (8), the headcounts of all the segments add up to m ($\sum_{i=1}^{q} S_i.hc = m$). In order to select *K* points from *Traj*, we come down to select smaller pieces of points from each segment according to associated $S.hc$. After point ranking step, we can rank each point in a segment by $p.w$ inferred by equation (9), then we select $S.hc$ points from segment *S* and add them to a points queue. Finally, all the points collected from different segments add up to *m*, thus we get the most important *m* points according to our TS algorithm. A formal description of this operation is demonstrated in Figure 11. In line 4, *SelectTopK* is a general algorithm which selects top *k* objects from *n* objects ($k < n$) and merges them to a segment. It executes in complexity $O(nlogk)$ using minimal/maximal heap, thus the complexity of *SelectPoints* is $O(nlogm)$.

Algorithm SelectPoints(S)

Input: a segment who is assigned with a headcount

Output: a simplified segment S' whose points number equals to $S.hc$.

1. $S' = \emptyset$
1. $HeadCount = S.hc;$ *//point headcount of segment*
2. $Points = S.points;$ *//all points belong to segment*
3. $S' = SelectTopK(Points, HeadCount));$ *//select top points*
4. **Return** $S';$

Figure 11: The procedure of selecting points

Since previous steps including Segmentation, Segment Ranking and Point weighting all have complexity *O(n)*, the last step Point Selection cost $O(nlogm)$, so the time complexity of TS is $O(nlogm)$.

***Example* 3.** Consider a trajectory *Traj* demonstrated in Figure 12 a). After segmentation, *Traj* is partition into 2 segments: a walking segment S_1 and a driving segment S_2. Average heading changes and distances of S_1 and S_2 are calculated and normalized: $S_1.w$ = *0.4*, $S_2.w$ = *0.6*. After each point is weighted, the normalized weights vector of points in S_1 and S_2 are *{1, 0.33, 0.21, 0.13, 0.45, 0.15, 0.78, 0.345, 0.32, 0.65, 0.7, 0.5}* and *{0.55, 0.33, 0.65, 0.76, 0.7, 0.15, 0.1, 0.65, 0.32, 1}*. At last, given headcounts *m*, *m* points can be selected by selected 40% of *m* points from S_1 and 60% of *m* points from S_2. For instance, if *m* = 10, then we choose the top 4 points from S_1 and top 6 points from S_2, they are $\{p_1, p_7, p_{11}, p_{10}\}$ and $\{p_{22}, p_{16}, p_{17}, p_{15}, p_{20}, p_{13}\}$. The simplified trajectory is shown in Figure 12 b).

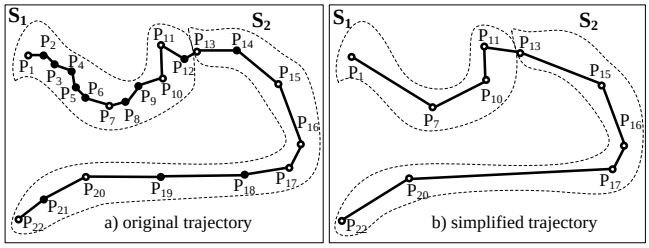

Figure 12: An example of TS algorithm

5. EXPERIMENTS

In this Section, we first present the experimental settings. Second, we introduce the evaluation approaches. Third, major results are reported followed by some discussions.

5.1 Settings

GPS devices: Figure 13 shows the GPS devices we chose to collect data. They are comprised of stand-alone GPS receivers (Magellan Explorist 210 and 300, G-Rays 2) and GPS phones. Users set these devices with different configurations, such as recording a GPS point every 5 seconds or every 10 meters.

GPS Data: We select 335 typical travel routes generated by 62 users in the past 2 years. Each trajectory contains at least one walk segment and one non-walk segment like driving, and the portion of walk segments exceeds 50 percents of the whole time span of the trajectory. Typically, people drive a relatively long way before reaching a real travel spot, e.g., mountains and lakes are usually located outside cities. Also, people prefer to spend more time in places of interest rather than driving on the way. For instance, users are more likely to spend 1 hour on the way while spending 5 or more hours on hiking in a mountain. Consequently, the distance of driving segment is usually larger than that people traveling (walking in most case) in a place of interest while the time spent for walking might be longer than driving time in a trip.

The total distance of these trajectories has exceeded 21,000 kilometers, and the distance of each trajectory is greater than 5 km. On average, each trajectory has 2100 points, a 63-kilometer distance and 4- hour time span.

Figure 13 GPS devices used in the experiments

Parameter Selection

τ of definition 5: Figure 14 shows how the average perpendicular deviation of each point in the original trajectory to the simplified

trajectory changes over τ. The three curves are results with different simplification rate. All three curves show that the average deviation is minimized when τ is set to 3. In other words, when τ is set to 3, the accumulated heading change can get the most out of the semantics meaning and skeleton information of the trajectory.

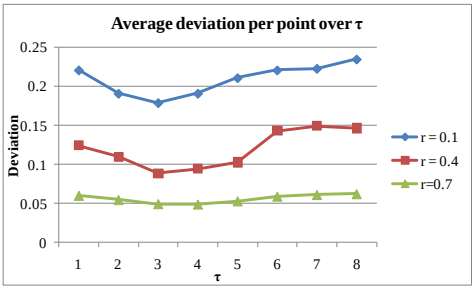

Figure 14 Selecting threshold τ for β

5.2 Evaluation Approaches

5.2.1 Evaluation criteria

Our proposed algorithms are evaluated both in terms of effectiveness of simplification result and efficiency of simplification execution.

Since one of the most important purposes of a trajectory is showing the reference knowledge to others, a relatively big deviation like 100 meters on a driving segment would not reduce the semantic meanings a trajectory implies, while a small deviation from the walk segment would loss the significant information about a trip

Based on this observation, in order to evaluate the average deviation of simplified trajectory, first we calculate the perpendicular distance from each raw point to simplified trajectory as deviation; second we normalize each deviation in walk segment and non-walk segment; finally we calculate the root mean square value of all the perpendicular distance. The result represents the average deviation for each point. Based on this, we propose two metrics to evaluate the effectiveness of our approach.

Average normalized perpendicular: we add up all the normalized perpendicular distance for all the points of all trajectories and calculate their root mean square value. The smaller the average deviation is, the better the algorithms is.

Correct Rate: we compare the simplified result for a single trajectory performed by DP and TS, and define another criterion named 'Correct Rate'. For either one of DP or TS, suppose it is TS. If TS outperforms DP by n_1 trajectories, but the left n_2 trajectories have bigger average deviation than DP, then:

$$Correct\ Rate\ of\ TS = \frac{n_1}{n_1 + n_2} \qquad (10)$$

While correct rate of DP is

$$Correct\ Rate\ of\ DP = \frac{n_2}{n_1 + n_2} \qquad (11)$$

5.2.2 Baseline

Since DP algorithm is wildly used for line simplification problem, and it's regarded as the most effective simplification algorithm till now, we take it as baseline.

5.3 Results

5.3.1 Effectiveness

Figure 15 shows the average normalized perpendicular deviation of TS and DP with different simplification rate. We can see that TS outperforms DP at any simplification rate.

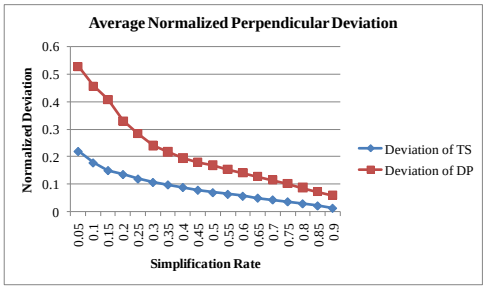

Figure 15 Average normalized perpendicular deviation of TS and DP

Figure 16 shows the correct rate for TS and DP with different simplification rate. When simplification rate is 0.05, TS has correct rate of 0.55 and correct rate of TS increases with the increase of simplification rate. So at any simplification rate, TS outperforms DP.

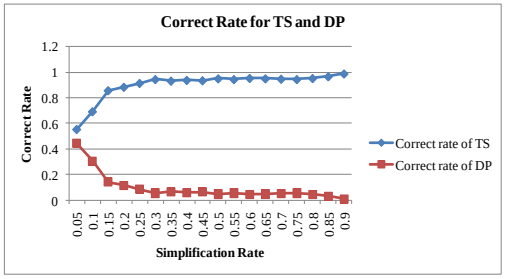

Figure 16 Correct Rate of TS and DP

Suppose the simplification r rate is normal distributed, $r \sim N(0.475, 0.0673)$, where mean value of r is 0.475 and variance of r is 0.0673. Based on this, the expected average normalized deviation of TS is calculated to be 0.078 and the expected average deviation of DP is 0.19. It means on average TS has only 41% deviation compared to DP. Similarly, based on normal distribution, we can draw that on average the correct rate of TS is 0.91 while that of DP is 0.09, which means TS is about 10 times better than DP for this criterion.

5.3.2 Efficiency

Since sometimes simplification is performed on line, speed is very important. DP adopts a divide and conquer method to scan each point in the trajectory. Suppose we need to maintain the most important m points. The average complexity of DP is $O(n * logn * logm)$, and in worst case it degenerates to $O(n^2 * logm)$. On the other hand, as analyzed in 4.4, both in average case and worst case, the complexity of TS is $O(n * logm)$. So in theory TS is faster than the DP.

As shown in Figure 17, TS outperforms DP in time cost. Based on normal distribution, on average TS is 4.5 times faster than DP.

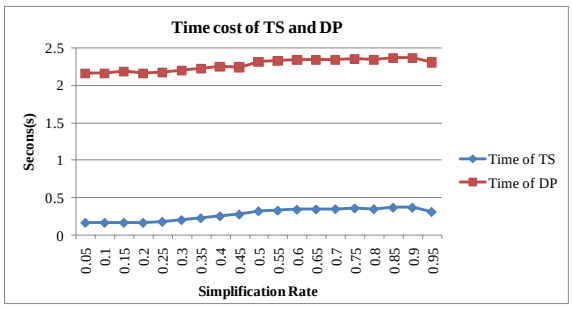

Figure 17 Time cost of TS and DP

5.4 Discussions

Experimental results show that TS outperforms DP both in effectiveness and efficiency. Comparing simplified results by DP and TS in Figure 2 and Figure 3, TS has worse performance in driving segment, while better in walking segment. According to Figure 15 and 16, we know that the better performance part outperforms the worse part. The reason why walking segment has better performance lies in two aspects: 1) segmentation weighting step considers the average heading change of segment, on which feature walking segment is more significant. 2) point weighting step also considers heading change, generally walking point will be weighted bigger on this feature since people tend to change heading directions while walking. Further, this method is robust to some dense noise point cluster, because according to Equation (1), the accumulated heading of a point in random generated noise cluster will be very small, hence won't be weight big.

Actually, in the process of segment weighting and point weighting, we are trying to balance the importance of overall skeleton and semantic meaning of a trajectory. By applying segment's distance and point's neighbor distance, the skeleton of trajectory is maintained. By applying segment's average neighbor heading change and point's heading change, the semantic meaning of the trajectory is maintained.

Our approach is proposed for trajectories that have at least 50% duration walking and all are longer than 5km, other cases may not outperform DP if the trajectory's walking part does not take up a proper duration proportion.

6. CONCLUSION

In this paper, we proposed a trajectory simplification algorithm, called TS, for location-based social networking services. This algorithm aims to maintain both the shape skeleton and semantic meanings of a trajectory when reducing redundant points from the trajectory. At the same time, the TS combine the strategies of local and global processing during line simplification. Therefore, with a better efficiency than the DP algorithm, TS is more effective in simplifying trajectories in LBSN systems where people expect to share and follow others travel routes. Two features consisting of the heading change and the neighbor distance are used to weight the importance of a point. We evaluate our TS algorithm using 335 trajectories collected by 62 users over 2 years and a new metric called normalized perpendicular distance. As a result, our method outperforms baseline with a 10-times Correct Rate beyond baseline. Meanwhile, according the mean normalized perpendicular distance, our method only has a 41% deviation of the baseline with a 4.5-times faster efficiency.

In the future, we intend to extend our work to better balance the skeleton and semantic meaning of a GPS trajectory. So far, when the point headcounts are extremely small, the skeleton information of trajectories processed by our approach could be worse than the results of the DP. More factors should be taken into account to determine the weights of segments.

7. REFERENCES

[1] Bikely: http://www.bikely.com/

[2] GPS Track route exchange forum: http://www.gpsxchange.com/

[3] GPS sharing: http://gpssharing.com/.

[4] Zheng, Y. et al. GoLife2.0: A Location-Based Social Networking Service. In proceedings of International Conference on Mobile Data Management 2009, IEEE Press: 211-212

[5] Douglas, D. and T. Peuker, 1973, Algorithms for the reduction of the number of points required to represent a digitised line or its caricature, The Canadian Cartographer, Vol 10, pp. 112-122.

[6] Zheng, Y., Zhang, L., Xie X., Ma, W. Y. Mining interesting locations and travel sequences from GPS trajectories for mobile users. In Proceeding of WWW2009, (Madrid, Spain. April 2009), ACM Press: 791-800.

[7] T. Lang, "Rules for Robot Draughtsmen", Geographical Magazine, Vol. XLII, No. 1, Oct. 1969, pp. 50-51.

[8] W. R. Tobler, Numerical Map Generalization, Michigan Inter-University Community of Mathematical Geographers, Discussion Paper No. 8, Department of Geography, University of Michigan, January 1966.

[9] Taylor, G. (2005) Line simplification algorithms, Retrieved 15 April 2005 http://www.comp.glam.ac.uk/pages/staff/getaylor/papers/lcwin.pdf

[10] McMaster, R.B., 1983, A mathematical evaluation of simplification algorithms, Proceedings, Sixth International Conference on Automated Cartography AutoCarto 6, pp. 267-276.

[11] Jenks, G.F., 1989, Geographic logic in line generalisation, Cartographica, Vol. 26, No. 1, pp. 27-42.

[12] Zheng, Y. et al. Understanding mobility based on GPS data. In Proc. Ubicomp'08, (Seoul Korea, Sept. 2008), ACM Press: 312-321.

[13] Experimental Cartographic Unit, Royal College of Art: Automatic Cartography and Planning,London, Architectural Press, 1971.

[14] Zheng, Y., Liu, L., Wang, L. Xie, X. Learning transportation modes from raw GPS data for geographic applications on the Web. In Proceedings of WWW 2008, (Beijing China, April 2008), ACM Press: 247-256.