Geoinformatica

DOTS: An online and near-optimal trajectory simplification algorithm

--Manuscript Draft--

Manuscript Number:	GEIN-D-16-00041
Full Title:	DOTS: An online and near-optimal trajectory simplification algorithm
Article Type:	Manuscript
Keywords:	Geographic information systems; Location based services; GPS; Trajectory simplification; Priority queue; Directed acyclic graph
Corresponding Author:	Weiquan Cao, Ph.D. The Key Laboratory of Science and Technology On Blind Signal Processing Chengdu, Sichuan CHINA
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	The Key Laboratory of Science and Technology On Blind Signal Processing
Corresponding Author's Secondary Institution:	
First Author:	Weiquan Cao, Ph.D.
First Author Secondary Information:	
Order of Authors:	Weiquan Cao, Ph.D.
	Yunzhao Li, Ph.D.
Order of Authors Secondary Information:	
Funding Information:	

Geoinformatica manuscript No.
(will be inserted by the editor)

DOTS: An online and near-optimal trajectory simplification algorithm

Weiquan Cao · Yunzhao Li

Received: date / Accepted: date

Abstract The last decade witnessed an increasing number of location acquisition equipments such as mobile phone, smart watch etc. Trajectory data is collected with such a high speed that the location based services (LBS) meet challenge to process and take advantage of that big data. Good trajectory simplification (TS) algorithm thus plays an important role for both LBS providers and users as it significantly reduces processing and response time by minimizing the trajectory size with acceptable precision loss. State of the art TS algorithms work in batch mode and are not suitable for streaming data. The online solutions, on the other hand, usually use some heuristics which won't hold the optimality. This paper proposed a **D**irected acyclic graph based **O**nline **T**rajectory **S**implification (DOTS) method which solves the problem by optimization. Time complexity of DOTS is $O(N^2/M)$. A cascaded version of DOTS with time complexity of $O(N)$ is also proposed. To our best knowledge, this is the first time that an optimal and online TS method is proposed. We find both the normal and cascaded DOTS outperform current TS methods like Douglas-Peucker[6], SQUISH[12] etc. with pretty big margin.

Keywords Geographic information systems · Location based services · GPS · Trajectory simplification · Priority queue · Directed acyclic graph

W. Cao
The Key Laboratory of Science and Technology On Blind Signal Processing
Chengdu 610041 P.R.China
E-mail: caoweiquan322@126.com

Y. Li
The Key Laboratory of Science and Technology On Blind Signal Processing
Chengdu 610041 P.R.China
E-mail: lyz_pengzhou@163.com

1 Introduction

LBS is becoming more and more popular around the world with assistance of many excellent front-end mobile applications. What's more, LBS has gradually broken the limitation of traditional applications such as e-map and navigation. The great value behind location data attracted many researchers' interest. It seems that every modern mobile application takes advantage of location data to serve its users better. E.g. Recent navigation applications are able to smartly detect traffic congestion and suggest you the fastest route. *Uber* will call the nearest driver for you. *Apple health* records and analyzes daily sports data to monitor your body condition and collect personal statistics for customized disease treatment. All back-end services behind these wonderful applications have to process large amount of trajectory data.

The speed that location data is being collected may beyond one's estimation. Nirvana et al. pointed out that 100 MB location data would be generated everyday if 400 devices collect location points with period of 10 seconds without any reduction[11]. Along with the popularity of LBS related applications, servers have to use big data utilities such as Spark, Hadoop etc. to analysis trajectories fast enough. TS algorithm aims to reduce size of trajectory data without losing much accuracy and thus helps a lot. Researchers find that this tradeoff is worthy. In most cases, we need to mine frequent motion patterns, period patterns etc. from trajectories. These data mining algorithms should be robust to slight changes of location points.

A trajectory, in general case, is a sequence of 3-tuple records. Each tuple consists of *longitude*, *latitude* and *timestamp* of a location point. We represent this as $p_i = (x_i, y_i, t_i)$ for short in subsequent sections. As described above, a trajectory T is denoted by

$$T = \{p_i | i = 1, 2, 3, ..., N\} \tag{1}$$

where N is $+\infty$ for a streaming trajectory.

A TS algorithm is meant to find an indices set $S = \{k_1, k_2, ..., k_M\}$ so that $T' = \{p_{k_i} | i = 1, 2, 3, ...M\}$ is an approximation of the original trajectory T. Note that S must be monotone increasing. Additionally, we always require $k_1 = 1$ to include the first point in the final approximation.

TS algorithm always introduces accuracy loss. In general, the fewer the number of simplified points is, the larger the accuracy loss would be. Thus there are straightforward two dual problems for TS algorithms.

The *min-#* problem aims to minimize the number of reduced points while maintain the specified approximation error ε. The *min-ε* problem, on the other hand, tries to minimize the approximation error with a specified number M of location points. These two targets are somewhat opposite. TS algorithms then need to find the best tradeoff between these two targets under some criterion.

Douglas-Peucker[6] might be the most well known polygonal approximation algorithm. The main idea of Douglas-Peucker is to move the point with the largest error to the simplified set S. This operation is repeated until no

point has error that exceeds the given threshold. Zheng et al. proposed another heuristics TS algorithm by maintaining both the shape skeleton and semantic meanings of a trajectory[4]. As illustrated in [4], this algorithm outperforms Douglas-Peucker under different compression rate. However, the author pointed out that the result might vary if evaluated on different datasets. Algorithm in [4] is not open source yet and is hard to reproduce due to its multiple parameters.

TS algorithms above lie in category of *batch mode*. That is, these algorithms need the entire trajectory being collected before doing any data reduction operations. This is acceptable for applications like animation curve approximation, sports data analysis etc. However, a great deal of applications call for TS algorithms that could deal with streaming data. Since portable devices have a limited storage capacity. It's usually impractical to collect the entire trajectory before it is compressed. The time delay would also be unacceptable for real time services if the trajectory is not uploaded to server until it's finished. One alternative is to simplify location data at server side. That would however give big burden to LBS servers and the transfer network. It's a pretty practical choice to reduce data *online* at capture devices.

The algorithms that is able to simplify streaming trajectory data is also called *online* algorithms. *Online* TS algorithms have become an active research area due to its wide use compared to *batch mode* TS algorithms. One trivial implementation is just to select location data with a predefined or random[13] interval i.e. the *k-th* simplification. This strategy does not make use of any spatio-temporal feature of the given trajectory and thus has a poor performance. The open window based algorithm called BOPW[9,11] is widely used, thanks to its easy implemention. BOPW approximates as more points as possible by a 2-point segment until the error exceeds the predefined threshold. The last point that holds a legal error would be output to set S and selected as the start point of the new open window. SQUISH, proposed by Muckell et al.[12], maintains a fixed window size β. When new point comes and the window size exceeds β, SQUISH removes the point with the least cost. This is done recursively and the window size will keep constant. One obvious drawback of SQUISH is that we can not limit the approximation error as the input trajectory keeps growing. Katsikouli et al. select local minima/maxima of curvature as candidates of the simplified trajectory and achieve an efficient $O(N)$ online TS algorithm[8]. In our experiments, we find that this method performs quite poor compared with modern TS algorithms despite of its efficiency.

All algorithms above are designed with heuristics. They all somewhat belong to greedy methods. As a result, there's no an algorithm that could outperform others under different data sets and different error measurements[4]. In contrary, an optimization based TS algorithm proposed by Imai H. et al.[7] could perform the best under quite a lot of error measurements. This algorithm, however, has a time complexity of $O(N^2)$ which is too expensive in practical use. The most calculation consuming operation of [7] is the *edge test* when constructing a directed acyclic graph (DAG). Daescu et al. tried to reduce complexity of [7] by avoiding unnecessary edge tests with an ef-

Table 1 Summary and comparison of popular trajectory simplification algorithms and the proposed DOTS method

Algorithm	Time cost	Mode	Design pattern	Temporal usage
D-P[6]	$O(N \log N)$	Batch	Heuristics	No
TS[4]	$O(N \log M)$	Batch	Heuristics	Yes
MRPA[3]	$O(N)$	Batch	Hybrid	Yes
SQUISH[12]	$O(N \log M)$	Online	Heuristics	Yes
PERSISTENCE[8]	$O(N)$	Online	Heuristics	No
DOTS	$O(N^2/M)$	Online	Optimization	Yes
DOTS-CASCADE	$O(N)$	Online	Hybrid	Yes

ficient priority queue data structure[5]. Kolesnikov et al. achieved the same goal by limiting search space of the edge tests[10]. Chen et al. proposed a fast and multi-resolution polygonal approximation method MRPA[3] to further improve the efficiency of [5,10] by multi-step approximation. Existing optimization based methods[7,5,10,3] all work in batch mode and thus could not adapt to streaming data.

Inspired by the idea of DAG[7,5,10], this paper proposed a novel TS algorithms *DOTS* that works in online mode while hold near-optimality. Though DOTS is much faster than [7], it's still relatively slower than several online heuristics e.g. BOPW[9,11], SQUISH[12], etc. We later find that it's feasible to use several DOTS simplifier concurrently as a pipeline and successfully achieve time complexity of $O(N)$. We name this as *DOTS-CASCADE* algorithm. Note that DOTS-CASCADE does not call for multi-core and works in one single thread. DOTS-CASCADE is fast enough while could still handle streaming trajectories. To our best knowledge, this is the first time that an online and optimization based TS algorithm is proposed. In our experiments in Sect. 4, we find these two algorithms outperform existing TS algorithms under multiple error measurements with big margin. The DOTS algorithm was implemented with QT/C++ and is now available for download through github[1].

We list several popular TS algorithms and show their differences of several aspects in Tab. 1. In Tab. 1, N denotes size of the original trajectory T, while M denotes size of the simplified trajectory $T^{'}$. For online TS algorithms, N and M no longer make sense but should be treated similarly as in batch mode.

This paper is organized as follows. We introduce the problem of trajectory simplification and several existing solutions in section 1. Related work such as error measurements, a brief introduction to DAG etc. is described in section 2. Implementation details of DOTS as well as a cascaded version are shown in section 3. The proposed algorithms are evaluated with dataset Mopsi[1] and GeoLife[16,14,15] in section 4. Section 5 concludes this paper.

[1] https://github.com/caoweiquan322/dots [2]

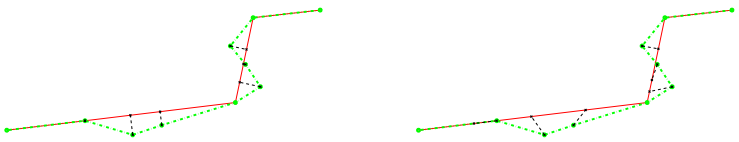

Fig. 1 The difference between PD and SED error, the left figure is PD and the right one is SED.

2 Related work

2.1 Error measurements

There are plenty of error measurements for trajectory simplification results. Suppose that i and j are consecutive elements of the simplified indices set S. Then we have $p_i \in T^{'}$ and $p_j \in T^{'}$. For any points $p_k \in T$ between p_i and p_j, Perpendicular distance (PD) accumulates the minimum distance from p_k to line segment $\overline{p_i p_j}$ as the error. PD is logically straightforward but lacks of time information. From the definition of trajectory (see Eq. 1) we notice that *timestamp* is a key field of each location point. As a result, researchers find the Synchronized Euclidean Distance (SED) a better measurement of approximation error. The difference between PD and SED is illustrated in Fig. 1. The original trajectory (green dash dotted) contains 10 location points in total while the simplified trajectory (red solid) contains only 4. The black dash lines denote PD and SED distance respectively. The SED error is mathematically defined as below:

$$SED_k = \sqrt{(x_k - x_k^{'})^2 + (y_k - y_k^{'})^2} \tag{2}$$

where

$$x_k^{'} = x_i + \frac{t_k - t_i}{t_j - t_i}(x_j - x_i) \tag{3}$$

$$y_k^{'} = y_i + \frac{t_k - t_i}{t_j - t_i}(y_j - y_i) \tag{4}$$

The virtual location point $p_k^{'}$ is called *time synchronized point* of p_k.

Though SED describes approximation error very well, we find it's difficult to accumulate consecutive SEDs between arbitrarily specified indices i and j quickly. The Squared SED (SSED), in contrary, could be calculated efficiently within $O(1)$ time[3]. Let P_i^j denote the sub-trajectory between indices i and j. If we approximate P_i^j with line segment $\overline{p_i p_j}$, then the accumulation of SSED for every points between i and j would be defined as Local Integral Square

Synchronized Euclidean Distance (LISSED)

$$\begin{aligned}
&LISSED_i^j \\
=&\sum_{i<k<j} SSED_k \\
=&\sum_{i<k<j} SED_k^2 \\
=&(c_1^2+c_3^2)(j-i-1)+(c_2^2+c_4^2)(S_{t^2}^{j-1}-S_{t^2}^i) \\
&+2(c_1c_2+c_3c_4)(S_t^{j-1}-S_t^i) \\
&+(S_{x^2}^{j-1}-S_{x^2}^i)+(S_{y^2}^{j-1}-S_{y^2}^i) \\
&-2c_1(S_x^{j-1}-S_x^i)-2c_3(S_y^{j-1}-S_y^i) \\
&-2c_2(S_{xt}^{j-1}-S_{xt}^i)-2c_4(S_{yt}^{j-1}-S_{yt}^i)
\end{aligned} \tag{5}$$

where

$$\begin{aligned}
c_1=\frac{x_it_j-x_jt_i}{t_j-t_i} \quad c_2=\frac{x_j-x_i}{t_j-t_i} \\
c_3=\frac{y_it_j-y_jt_i}{t_j-t_i} \quad c_4=\frac{y_j-y_i}{t_j-t_i}
\end{aligned} \tag{6}$$

S_*^n denotes the accumulation of corresponding variable $*$ indexed from 1 to n. The accumulation could be stored and updated once a new location point is input. Thus the time complexity of calculating LISSED is $O(1)$. Refer to [3] about detailed derivation of Eq. 5. If we sum all the LISSEDs, we get Integral Square Synchronized Euclidean Distance (ISSED) of the simplified trajectory $T^{'}$

$$ISSED=\sum_{p_{k_i}\in T'} LISSED_{k_i}^{k_{i+1}} \tag{7}$$

2.2 Approximation error and compression rate

In general, trajectory simplification decreases the number of points and introduces approximation error at the same time. Here let r denote the compression rate of a TS algorithm

$$r=N/M \tag{8}$$

where N denotes size of the original trajectory T, while M denotes size of the simplified trajectory $T^{'}$. For online TS algorithms, N and M no longer make sense but should be treated similarly as in batch mode.

The approximation error tends to be larger as r becomes larger. For the *min-#* problem, we fix error and maximize r. While for the *min-ε* problem, we fix r and minimize the approximation error. Thus it's not reasonable to compare only approximation error or compression rate when evaluating a TS method. Instead we illustrate the e/r curve, i.e. the relationship between approximation error and compression rate when they vary, to show effectiveness of the simplification.

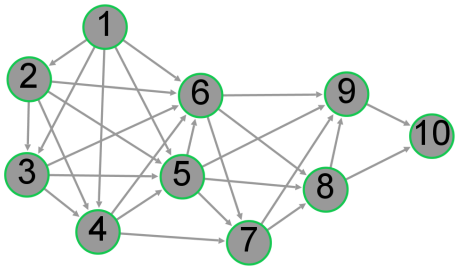

Fig. 2 The DAG that corresponds to Fig. 1

2.3 An introduction to DAG

DAG, i.e. directed acyclic graph, could be used to describe all potential simplified trajectories of a trajectory T with specified threshold of LISSED[7]. A DAG is made up of two sets: the vertices set V and the edges set E.

V contains all the location points of T. Any two vertices of V have a directed edge if the LISSED between them is less than the threshold ε.

$$E = \{(p_i, p_j) | i < j \ and \ LISSED_i^j < \varepsilon\} \tag{9}$$

Note that elements of E are all directed. That is, we can only have the vertices with smaller indices point to the ones with larger indices.

Having the DAG constructed, every path from p_1 to p_N within the graph corresponds to a feasible approximation of trajectory T. Then the *min-#* problem could be solved in batch mode by finding the shortest one. Obviously, DAG based method could find the optimal solution to the *min-#* problem. However, constructing DAG is time consuming. We have to test every pair (p_i, p_j) to see if they belong to E. This operation is called *edge test*. It takes $N(N-1)/2$ edge tests to determine the edge set E of the DAG. Thus the time complexity is $O(N^2)$. This is generally unacceptable in practical use.

The DAG that corresponds to Fig. 1 is shown by Fig. 2. As we see, one of the shortest path (1, 5, 9, 10) is selected as the simplified trajectory in Fig. 1. However, there're other paths of the same length that satisfy the approximation error threshold ε, e.g. path (1, 5, 8, 10), (1, 6, 9, 10) etc. This indicates that there's multiple solutions to the *min-#* problem and there's still room for finding the *best* one.

Besides, DAG based method requires the whole trajectory being input before outputing the optimized approximation. That's why previous DAG based methods could not adapt to online mode. To design a DAG based Online Trajectory Simplification (DOTS) algorithm, we need finding a way to determine part of the shortest path before the whole DAG is completely constructed.

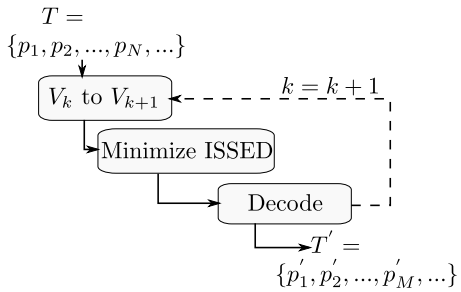

Fig. 3 An overview of DOTS

3 Algorithm implementation

3.1 Overview

The work flow of DOTS algorithm is illustrated by Fig. 3. DOTS is implemented in recursive manner. At each iteration, we first add a new layer to the incremental DAG efficiently which would be described in Sect. 3.2. Then the ISSED will be locally minimized by adjusting edges between consecutive layers. Finally we keep monitoring the *alive* status of vertices and perform viterbi-like decoding on the incremental DAG.

Since the developed DOTS is of $O(N^2/M)$ time complexity. We further proposed a cascaded version of DOTS which is able to simplify trajectories as fast as $O(N)$. This will be described in Sect. 3.6.

3.2 Efficient and incremental DAG construction

We need to construct DAG efficiently such that it won't cost too much time and could further adapt to streaming trajectory data. Dynamic Programming (DP) is frequently used to find the shortest path between two vertices of an arbitrary graph. Once the DAG is constructed, the remaining work is just to run a general DP algorithm from p_1 to p_N on the graph. However, for this specific problem, we could actually implement DP early during construction of DAG.

Let $L^{\varepsilon}(P_j)$ denote the minimum length of path from p_1 to p_j with respect to tolerance ε. Then $L^{\varepsilon}(P_j)$ could be determined recursively as below

$$L^{\varepsilon}(P_j) = \min_{i<j \text{ and } LISSED_i^j<\varepsilon} L^{\varepsilon}(P_i) + 1 \tag{10}$$

From the definition of Eq. 10, we notice that there's at most one parent for any vertice p_j. Thus the DAG retrogrades to a tree structure. $L^{\varepsilon}(P_j)$ then represents the *layer* or *depth* of vertice p_j in the tree. The root of the tree is always $\{p_1\}$. This is demonstrated in Fig. 4.

If we are solving the *min-#* problem in batch mode, then it's easy to find the solution by travel through the unique path from p_N to p_1 inversely. This

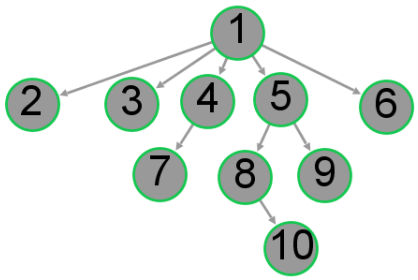

Fig. 4 DAG that is simplified as a tree

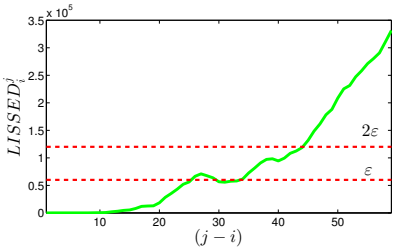

Fig. 5 The growing trend of $LISSED_i^j$ with respect to $(j-i)$

could be done within $O(M)$ time. If the method need adapting to online mode, then things become a little complex.

One important clue that inspired us is that each layer of the tree must be added without feeding the entire trajectory. Otherwise the TS algorithm is not *online* any longer. Let's assume that the first k layers of the tree has already been completely constructed. Then some termination criterion is necessary to construct the $k+1$ layer with finite number of new location points.

It's observed from Fig. 5 that the local approximation error $LISSED_i^j$ is growing with respect to the distance of location points, i.e. $(j-i)$. This is general among various of natural trajectories. Thus it's reasonable to assume that no j would satisfy the *edge test*, i.e. Eq. 9, if p_j is sufficiently far away from p_i.

Then the question is: *how far away is enough?* We find it's just OK to simply enlarge the error threshold ε by a linear factor η. Once Eq. 11 were met, any location points after p_{j^*} would be omitted. Proper value of η is in range $(1, +\infty]$. This paper used $\eta = 2.0$ for experiments in Sect. 4. We find the TS result is not sensitive to different η values.

$$j^* = \arg\min_j LISSED_i^j > \eta\varepsilon \tag{11}$$

The DAG is then increamentally constructed as described in Tab. 2. Let V_k denote the points that have the same layer, i.e. $L^\varepsilon = k$. Note that V_1 is always initialized as $\{p_1\}$. The termination set $Term_k$ is maintained in iteration k to

Table 2 Incremental DAG construction (itr_k)

0	We have got $V_1, V_2, ..., V_k$
1	$V_{k+1} = \Phi$; $Term_k = \Phi$
2	WHILE p_j << TRAJECTORY_DATA_STREAM
3	FOR p_i in V_k
4	IF $p_i \in Term_k$
5	CONTINUE
6	ENDIF
7	IF $LISSED_i^j < \varepsilon$
8	$V_{k+1} = V_{k+1} \bigcup \{p_j\}$
9	$Parent(p_j) = p_i$
10	BREAK FOR
11	ELSEIF $LISSED_i^j > \eta\varepsilon$
12	$Term_k = Term_k \bigcup \{p_i\}$
13	ENDIF
14	ENDFOR
15	IF $Term_k == V_k$
16	BREAK WHILE
17	ENDIF
18	ENDWHILE
19	Minimize ISSED locally according to Sect. 3.3 (*)
20	Try to decode and output according to Sect. 3.4

check if we need input more location data or not. V_{k+1} is finally determined when the outside *while* loop is broken in line 16.

The difference between [3,5,10] and our DAG construction is that we update the DAG layer by layer incrementally and we do not need the entire trajectory to be collected when constructing preceding part of the whole DAG. This makes it possible to reduce streaming trajectories online with acceptable delay.

3.3 Locally minimize ISSED

ISSED from p_1 to each element of V_k is maintained as α_p during each iteration. This is done by updating ISSED of each point based on its parent.

$$\alpha_{p_j} = \alpha_{Parent(p_j)} + LISSED_{Parent(p_j)}^{p_j} \tag{12}$$

The ISSED α_{p_j} could further decrease if we choose parent of p_j carefully from among previous layer. Suppose $p_j \in V_{k+1}$ and there're multiple points in V_k that satisfy Eq. 9 with p_j. Let's choose the optimal one as the parent of p_j as shown in Eq. 13.

$$Parent^*(p_j) = \underset{p_i \in V_k \ LISSED_i^j < \varepsilon}{\arg\min} \alpha_{p_i} + LISSED_i^j \tag{13}$$

The parent tuning of an example point p_j is illustrated in Fig. 6. Though we assigned one parent for each element of V_{k+1} in line 9 of Tab. 2, we need

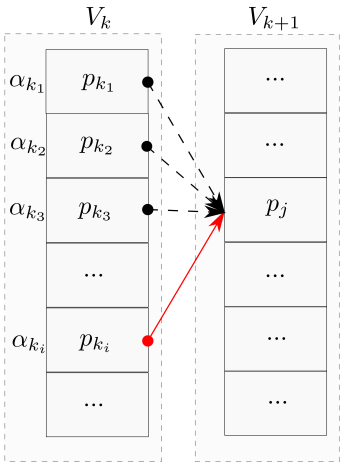

Fig. 6 Locally minimize the ISSED by tuning parent nodes

still tuning their parents here. Note that this operation, i.e. line 19 of Tab. 2, is optional since it makes no difference on the result of *min-#* problem. This just makes the approximation error even less while keep compression rate r unchanged.

3.4 Decode and output

Let's extend the definition of $Parent()$ as $Ancestor()$. For a location points set S,

$$\begin{aligned} Ancestor^1(S) &= \{Parent(p) | \forall p \in S\} \\ Ancestor^m(S) &= Ancestor^1(Ancestor^{m-1}(S)) \end{aligned} \tag{14}$$

Thus for elements in layer k of the DAG tree, i.e. V_k, $Ancestor^m(V_k)$ would denote all the elements in layer $(k-m)$ that still have their children in layer k. We call these elements *alive* vertices. While others are called *dead* vertices. This is illustrated in Fig. 7. Each time we construct a new layer V_{k+1}, we need to update *alive* status of previous layers. *Alive* vertices are marked light and *dead* ones are marked dark as shown in Fig. 7.

It's observed that the number of *alive* elements in V_{k-m} decreases quickly with respect to $(k-m)$. This result comes from a wide test on natural GPS trajectory datasets. One sample from Mopsi dataset[1] is shown in Fig. 8. We find the relationship between expectation of *alive* vertices number E_{k-m} and the distance $(k-m)$ could be well modeled as Eq. 15. Curve fitting based on Eq. 15 was done against the experimental result in Fig. 8 where the optimal parameter set (A, τ) is $(632.6, 0.4308)$. Goodness of fitting is shown in Tab. 3.

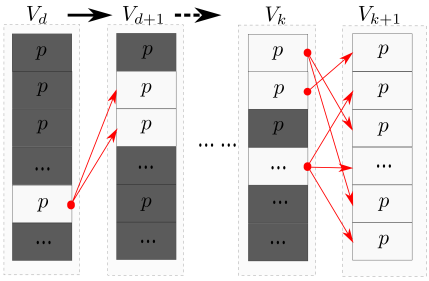

Fig. 7 Decoding procedure of DOTS algorithm

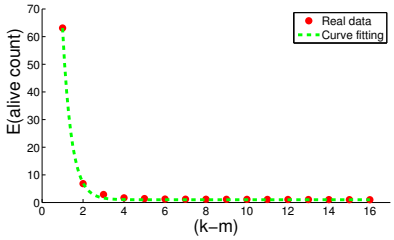

Fig. 8 Expectation of *alive* vertices number in V_{k-m} with respect to $(k-m)$

Table 3 Goodness of fitting Fig. 8 using Eq. 15

Goodness measure	Value
SSE	2.511
R^2	0.9993
Adjusted R^2	0.9992
RMSE	0.4235

Thus we may expect that layer V_d which has only one *alive* vertex in Fig. 7 would not be far away from V_{k+1}.

$$E_{k-m} = 1 + Ae^{-\frac{k-m}{\tau}} \tag{15}$$

We will keep monitoring and updating *alive* status of each layer. Tab. 4 lists the pseudo code of this section. DOTS updates *alive* status of V_d to V_k as long as the new layer V_{k+1} is constructed. Then it checks if V_d and layers afterwards contains only 1 single alive vertex. Those 1-alive-element layers will decode and output that single vertex to the simplified trajectory $T^{'}$ step by step. We see that DOTS could keep outputing new location points to $T^{'}$ and is an online TS algorithm.

During the decoding procedure, each location point needs looking back upon previous layers between V_d and V_k to maintain the *alive* status. Number of these operations is bounded by $O(\log N/M)$ since $Ancestor^m(S)$'s size exponentially decay to 1 with respect to m as discussed above (Eq. 15). Thus the time complexity of decoding procedure is $O(N \log N/M)$. Note that this

Table 4 The decoding and output procedure (itr_k)

0	We have just updated V_{k+1}
1	FOR $m = k : -1 : d$
2	FOR p_j in V_{m+1} and p_j is *alive*
3	Set $Parent(p_j)$ as alive
4	ENDFOR
5	ENDFOR
6	$m = d$
7	WHILE $m \leq k$ and V_m has 1 single *alive* vertex p^{m^*}
8	Output p^{m^*} to $T^{'}$
9	$m = m + 1$
10	ENDWHILE
11	$d = m$

conclusion comes partially from experiments and we will further give some proof to this in future works.

3.5 Complexity analysis

Time complexity of DOTS algorithm is mainly caused by *edge test*. Thus step 0 to 19 of Tab. 2, i.e. DAG construction and ISSED minimization, constitute the most time consuming part of DOTS. Number of *edge tests* is bounded by $O((N/M)^2 M)$ where N/M denote the average size of each layer and M denote the number of DAG layers.

The decoding procedure costs $O(N \log N/M)$ time as discussed in Sect. 3.4. So the total time complexity of DOTS is bounded by $O(N^2/M)$ or $O(Nr)$ equivalently where r denotes the compression rate (Eq. 8).

3.6 Cascaded version

Since modern online TS algorithms like SQUISH[12], PERSISTENCE[8] etc. are as fast as $O(N \log M)$ or even $O(N)$. It's necessary to boost efficiency of our DOTS algorithm. We noticed that time complexity of DOTS is bounded by $O(N^2/M)$. Along with inspiration of geometric progression, we designed a cascaded version of DOTS.

As shown in Fig. 9. DOTS-CASCADE is made up of several DOTS modules which are cascaded as a pipeline. Each DOTS module compresses its input sequence of points by a factor of r_0. After N points being input to the heading DOTS module, the k-th module would output about N/r_0^k points which will act as input of module $(k + 1)$. Time complexity of the k-th DOTS module would be $O(N/r_0^{k-2})$, respectively. We could then accumulate the total time complexity as below

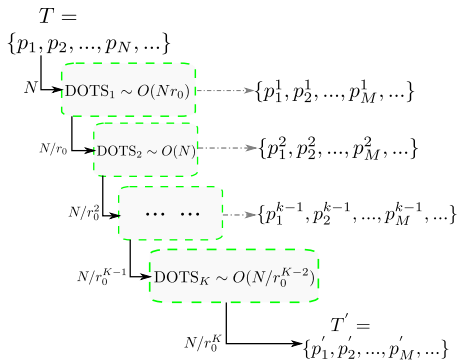

Fig. 9 The cascaded version of DOTS

$$\begin{aligned} T_{cascade} &\sim \sum_{k \in [1,\ K]} O(N/r_0^{k-2}) \\ &\sim \sum_{k \in [1,\ +\infty]} O(N/r_0^{k-2}) \\ &\sim O(Nr_0) \\ &\sim O(N) \end{aligned} \tag{16}$$

Thus we achieve an efficient $O(N)$ TS method while keep advantage of DOTS's good performance. For specified target compression rate r and compression rate of each DOTS module r_0, number of DOTS modules of DOTS-CASCADE is determined by

$$K = \lceil \log_{r_0} r \rceil \tag{17}$$

Note that multiple TS modules may introduce greedy effects on the final approximated trajectory. The number of output location points may not be as few as that of DOTS with the same error tolerance ε. As a result, DOTS-CASCADE is somewhat a hybrid of optimization and heuristics. The larger r_0 is, the more greedy our DOTS-CASCADE method is, vice versa.

In practical use, it's not convenient to control compression rate of each DOTS module. We could control the error tolerance ε instead. A monotone increasing sequence $\{\varepsilon_1, \varepsilon_2, ..., \varepsilon_K = \varepsilon\}$ is used in cascaded DOTS modules. A naive setting $\varepsilon_1 = 1000.0$ is used for all experiments of Sect. 4 as 1000.0 is quite a little value among all feasible ε. Other elements of the ε sequence could be determined as a geometric progression.

3.7 Online & batch mode

The proposed DOTS and DOTS-CASCADE algorithm could also be used in batch mode. We can simply input the whole trajectory T and collect all the

Table 5 Error measurements for evaluation

Measurement	Description
SED_{avg}	Average SED error of location points
SED_{mean}	Mean SED error of location points
$\text{SED}_{\max}$	Maximum SED error of location points
$\text{LISSED}_{\max}$	Maximum LISSED of simplification
ISSED_{avg}	Average value of the total ISSED

output as the simplified trajectory T'. In that case, our algorithm retrogrades to something similar to MRPA[3]. This enlightens the relationship between online and batched TS algorithms. On one hand, any online TS algorithms could be treated as batched. On the other hand, those batched TS algorithms could be regarded as online TS algorithms with infinite delay when processing streaming location data. Though the algorithm would become useless when delay approaches to infinity.

4 Experimental results

We take advantage of Mopsi[1] dataset and GeoLife V1.3[16,14,15] to evaluate performance of our DOTS & DOTS-CASCADE algorithm with existing ones. Mopsi contains 344 trajectories that were generated in 2011 autumn. Geolife was collected by 182 users in a period of over five years and contains 17,621 trajectories. These datasets have been widely used by location data researchers and thus could be used to compare performance of different algorithms effectively.

4.1 Error comparison

It makes no sense to evaluate approximation error or compression rate of a TS method separately. As discussed in Sect. 2.2, we would compare the e/r curve of TS methods. Multiple error measurements e.g. SED_{avg}, SED_{mean}, ISSED_{avg} etc. are used in experiments. See Tab. 5 for details.

We find DOTS and DOTS-CASCADE outperform existing TS algorithms a lot as illustrated in Fig. 10. DOTS achieves the best e/r curves under various of error measurements. The cascaded version sacrifices a bit performance by greatly improving the speed. We also find that our algorithms' superiority upon existing methods is not sensitive to different trajectory datasets. The result is similar despite of trajectory diversity such as multiple transportation modes, capture devices with different accuracy etc. The main cause of the result is that methods like Douglas-Peucker[6], PERSISTENCE[8], SQUISH[12] use heuristics as design pattern. While our DOTS and DOTS-CASCADE take advantage of optimization instead.

Despite the fact that the proposed methods outperform modern online TS algorithms, the simplification result of DOTS is even better than traditional

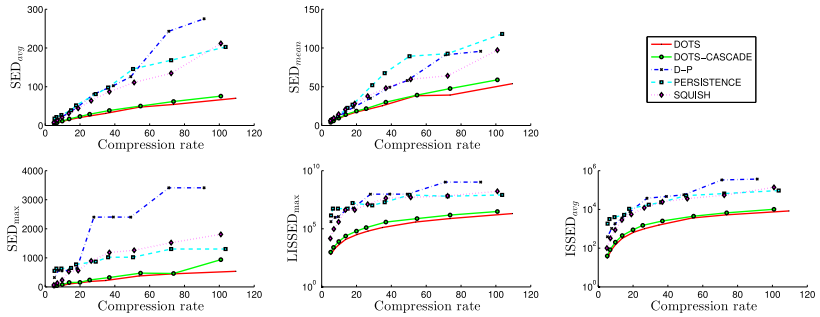

Fig. 10 e/r curves of the proposed DOTS&DOTS-CASCADE and existing TS methods under different error measurements

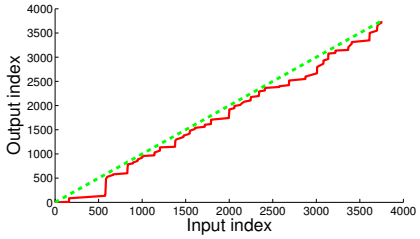

Fig. 11 Uncertain delay of DOTS algorithm

batched TS algorithms, e.g. Douglas-Peucker method. It's generally approved that batched TS algorithms approximate the original trajectory better than online methods with specified compression rate. The proposed DOTS methods, however, show that online TS algorithms could also have perfect accuracy with proper design paradigm.

4.2 Delay analysis

Both DOTS and DOTS-CASCADE have uncertain delay. This uncertainty is introduced by incremental DAG construction and the final viterbi-like decoding procedure. We evaluate DOTS delay of a sample trajectory from Mopsi with $\varepsilon = 4000.0$ in Fig. 11. The red solid line shows that simplified point indices are output as more and more location data being input to DOTS module. The green dashed is the reference. Ideally the red line would overlap the green dashed in which case there's no delay at all. For our practical result as shown in Fig. 11, the red line is always below the green dashed. The larger the gap between them is, the larger the delay is.

We also analyze the relationship between delay and compression rate r which is illustrated in Fig. 12. Generally, DOTS's delay becomes larger as r increases. However their ratio keeps quite stable. DOTS delay is practically about 3∼5 times of r.

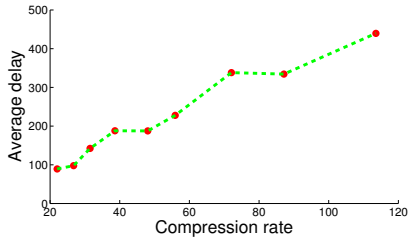

Fig. 12 Relationship between compression rate and DOTS delay

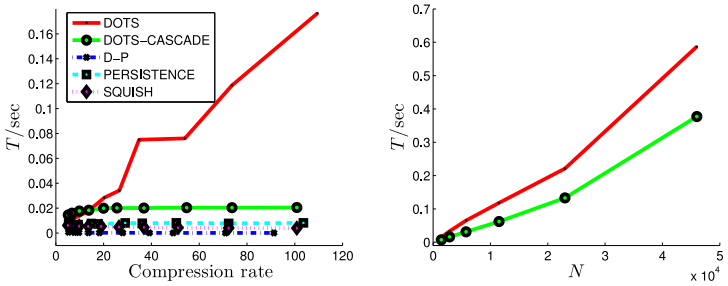

Fig. 13 A comparison on time cost of TS algorithms

As discussed in Sect. 3.4, the distance between V_d and V_k is bounded by $O(\log N/M)$ or $O(\log r)$ equivalently. Thus delay would be finite and be in proportion to $r \log r$ in theory.

4.3 Time cost

We further verify the efficiency of the proposed methods as shown in Fig. 13. With specified trajectory size N, time cost of all TS methods except DOTS is nearly constant regardless of different compression rate. DOTS has time complexity of $O(Nr)$ and thus its time cost increases in proportion to r.

Fig. 13(a) together with Fig. 13(b) help to confirm the theoretical analysis on time complexity of DOTS and DOTS-CASCADE which we mentioned in Sect. 3.5.

We would argue here that Douglas-Peucker[6] works so fast partially because it works in batch mode. There's relatively less logical branches in the workflow and thus it's easier to improve its efficiency by compiling options and programming skills. While it's much more difficult to optimize source code of online algorithms like DOTS. However the main bottleneck of batched TS algorithms lies in their unacceptable delay when processing streaming data or those offline trajectories with extremely large size. Previous online TS algorithms do not suffer from big complexity because most of them rely on heuristics and thus the rationale is usually quite simple compared with optimization based algorithms.

5 Concolusion

This paper proposed two novel optimization based trajectory simplification methods that both work in online mode. Both DOTS and DOTS-CASCADE outperform existing online and even batched TS algorithms a lot under various of error measurement. The DOTS method is a little slow compared with PERSISTENCE[8] etc. While the cascaded version is much faster and achieves time complexity of $O(N)$.

Future works include giving strict proof on the upper bound of DOTS's uncertain delay, developing online and near-optimal method that solves the *min-*ε problem etc.

Acknowledgements We'd like to thank editors and reviewers for their valuable comments on this paper.

References

1. Mopsi project (2015). URL http://cs.joensuu.fi/mopsi/
2. Cao, W.: Dots algorithm implemented with qt (2015). URL https://github.com/caoweiquan322/dots
3. Chen, M., Xu, M., Fränti, P.: A fast multiresolution polygonal approximation algorithm for gps trajectory simplification. Image Processing, IEEE Transactions on **21**(5), 2770–2785 (2012)
4. Chen, Y., Jiang, K., Zheng, Y., Li, C., Yu, N.: Trajectory simplification method for location-based social networking services. In: Proceedings of the 2009 International Workshop on Location Based Social Networks, pp. 33–40. ACM (2009)
5. Daescu, O., Mi, N.: Polygonal chain approximation: a query based approach. Computational Geometry **30**(1), 41–58 (2005)
6. Douglas, D.H., Peucker, T.K.: 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 (1973)
7. Imai, H., Iri, M.: Polygonal approximations of a curve-formulations and algorithms. Computational Morphology pp. 71–86 (1988)
8. Katsikouli, P., Sarkar, R., Gao, J.: Persistence based online signal and trajectory simplification for mobile devices. In: Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 371–380. ACM (2014)
9. Keogh, E., Chu, S., Hart, D., Pazzani, M.: An online algorithm for segmenting time series. In: Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on, pp. 289–296. IEEE (2001)
10. Kolesnikov, A., Fränti, P.: A fast near-optimal min-# polygonal approximation of digitized curves. In: Proc. IASTED Int. Conf. on Automation, Control and Information Technology-ACIT02, pp. 418–422 (2002)
11. Meratnia, N., Rolf, A.: Spatiotemporal compression techniques for moving point objects. In: Advances in Database Technology-EDBT 2004, pp. 765–782. Springer (2004)
12. Muckell, J., Hwang, J.H., Patil, V., Lawson, C.T., Ping, F., Ravi, S.: Squish: an online approach for gps trajectory compression. In: Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications, p. 13. ACM (2011)
13. Vitter, J.S.: Random sampling with a reservoir. ACM Transactions on Mathematical Software (TOMS) **11**(1), 37–57 (1985)
14. Zheng, Y., Li, Q., Chen, Y., Xie, X., Ma, W.Y.: Understanding mobility based on gps data. In: Proceedings of the 10th international conference on Ubiquitous computing, pp. 312–321. ACM (2008)

15. Zheng, Y., Xie, X., Ma, W.Y.: Geolife: A collaborative social networking service among user, location and trajectory. IEEE Data Eng. Bull. **33**(2), 32–39 (2010)
16. Zheng, Y., Zhang, L., Xie, X., Ma, W.Y.: Mining interesting locations and travel sequences from gps trajectories. In: Proceedings of the 18th international conference on World wide web, pp. 791–800. ACM (2009)

Weiquan Cao received the M.S. degree from the key laboratory of science and technology on blind signal processing, China in 2013 and is now a candidate for doctor's degree on communication and information system. His research interests include image steganalysis, graphics and location data mining.

Yunzhao Li received the doctor's degree from National University of Defense Technology, China in 2009. He's now a senior engineer of the key laboratory of science and technology on blind signal processing, China. His research interests include high performance computing, hardware acceleration and deep packet inspection.

Author Photographs

Click here to download Author Photographs Weiquan Cao.jpg

Author Photographs

Click here to download Author Photographs Yunzhao Li.jpg