An Online Data Compression Algorithm for Trajectories (An OLDCAT)

Ting Wang, *Senior Member, IACSIT*

***Abstract*—In the regime of "Big Data", data compression techniques take crucial part in preparation phase of data analysis. It is challenging because statistical properties and other characteristics need to be preserved while the size of data need to be reduced. In particular, to compress trajectory data, movement status (such as position, direction, and speed etc.) need to be retained. Moreover, for the increasing demand of real-time processing capability, "online" algorithms are becoming more desirable in data analysis. In this paper, we introduce an *On-Line Data Compression Algorithms for Trajectories (OLDCAT)*, which is an elegant, fast algorithm to effectively compress trajectory data to desirable volume. It is able to deal with real-time data, and scalable to adapt to different sensitivity, accuracy, and compression requirements. An evaluation of its parameter settings and a case study are also discussed in this paper.**

***Index Terms*—Data compression, trajectory data, online algorithm.**

I. Introduction

Over the last few decades, with the increasingly accurate positioning services (*e.g.* GPS, AIS, Mobile Phone Triangulation, RFID/Wi-Fi tracking etc.) and the decreasing price of their deployment, locational data becoming pervasive in our daily lives and scientific researches. Either indoor or outdoor, it is not difficult to obtain the trace, the velocity, and even the acceleration of any moving entity (referred to as an object in this paper) of our interest, providing proper equipment and infrastructure. Massive data have been collected in various research projects since early 90s [1]. As part of the "big data regime", interests in locational data have recently grown even more rapidly thanks to the new database technology and data mining techniques. When locational data coupled with time-stamps, it becomes spatial-temporal data — with both space (spatial) and time (temporal) information [2]. The timely sequence locations of an object define its trajectory. When multiple objects are concerned, an ID string or number is used in trajectory data for identifying the objects.

Trajectories of objects are widely used in a variety of applications, such as traffic modeling and supply chain management [3]. These efforts are being hampered by the sparse nature of data collection strategies, the sheer volume of the data, and technical issues associated with the use of the data. The enormous volume of data can easily overwhelm human analysis. This motivates the need for automated methods to compress and analyze the data.

There are mainly two different approaches in data compression:

1) Coding based: data is *encoded* using fewer number of bits than the original representation [4]. Data will be decoded before used in other applications or analysis;
2) Sampling based: part of the data is taken as *samples* while others are disregarded. Sampled data could be directly used in applications or analysis. However, good sampling algorithms will be needed to preserve statistical properties or other characteristics of the data.

While the coding approach is more relevant to data storage and presentations techniques (such as video and audio compression and streaming), sampling based compression is more commonly used with data obtained from measurements or signals, such as trajectory data. In this paper, we will focus on the sampling based approach, *i.e.* on how to sample trajectory data efficiently and effectively.

In computer science, an online algorithm [5] is one that can process its input piece-by-piece in a serial fashion, *i.e.*, in the order that the input is fed to the algorithm, without having the entire input available from the start. In contrast, an offline algorithm is given the whole problem data from the beginning and is required to output an answer which solves the problem at hand. In the context of trajectory data mining, online algorithms are extremely useful to enable real time configurations and optimizations. For example, to use the GPS data of the vehicles in traffic management, online algorithms should be preferred, as we could not wait till the end of the day to study the traffic pattern. Congestion and accidents would only be prevented or reduced if we were able to make online (real-time) adjustment to the policies (*e.g.* traffic lights, route recommendations). Data compression, as part of the data mining process, will also need to be online to deal with real-time, streaming data. Moreover, simple, fast yet elegant algorithms will be desirable so that more time and computational resources could be used on more tedious tasks (such as the classification and optimization problems).

In this paper, we propose an *On-Line Data Compression Algorithm for Trajectories (OLDCAT)*. It deals with the trajectory sampling and compression problem; identifies key points in a trajectory that define its characters; and re-constructs the trajectory with much fewer data points. A brief introduction to the existing works and challenges are discussed in Section II, followed by our discussion of the OLDCAT algorithm in Section III. We also show that by adjusting the parameters of OLDCAT, accuracy and scale of the compression is flexible and predictable in Section IV. A

Manuscript received April 12, 2013; revised June 25, 2013. This work was supported in part by the Singapore Economic Development Board (EDB).

Wang Ting is with SAP Asia Pte Ltd, Singapore (e-mail: dean.wang@sap.com).

case study of how OLDCAT could be used with some harbor data is discussed in Section V. Section VI concludes the paper with our findings and contributions.

II. Existing Works and Challenges

Based on dynamic programming, Bellman's algorithm [6], [7] has over 50 years of history in using point samples to represent lines and curves. Its results have been proven optimal with minimal root mean square errors under given conditions. The original implementation of this algorithm has a worst-case running time of O (O^3), where n is the number of points in the trajectory. This is a serious drawback that stops it being used with large data set or when real-time solution is required. A recent update [8] shows with additional storage, the running time of Bellman's algorithm could be reduced to O (O^2). However, it still not fit to the "online" requirement, and not able to deal with really "big" data.

The Douglas-Peucker algorithm [9] is the all-time classic line generalization heuristic. It is commonly used to represent a curve with a series of line segments, and thereby compress the storage requirement. However, as pointed out by Meratnia and de By in [10], Douglas-Peucker algorithm and its other implementations are not suitable for trajectory data as they could not deal with the time information, and may have treat the locational data in wrong sequence. Meratnian and de By thus proposed their own algorithm namely Top-Down Time Ratio (TD-TR) algorithm, which utilizes the temporal component in the trajectory data. Based on TD-TR, they extended their works to OPen Window Time Ratio (OPW-TR) and Open Window SPatial-temporal (OPW-SP) algorithms, which are similar to TD-TR but use different windowing criteria and error-thresholds. The major drawback of this group of algorithms is that they require all the trajectory data be available before choosing the line segments that represent the trajectory. Therefore they are "offline" solutions to the problem we are looking at.

The STTrace algorithm [11] is capable of dealing with online trajectory data streams, but it also requires object heading and speed information to characterize a trace. Similarly, the Semantic Trajectory Compression [12] algorithm uses urban map together with GPS data. The additional information (e.g. speed, heading, or map) may not be always available with the trajectory data, and thus the suitable scopes of these algorithms are limited. In our work, we aim to compress the trajectory with minimum information, i.e. the object ID, the time stamp and the longitude/latitude values from any positioning service.

Another challenge in trajectory data compression is that due to signal interference or technical flaw, data may not be accurate. For example, it's normal to have ~10m error in GPS data. For mobile phone triangulation, error can go as high as 1 mile when base stations are scarcely located. Even in a tiny city such as Singapore, the error is ~2m on average. Effective compression algorithms need to be able to deal with such error/noise in the raw trajectory data.

Moreover, scalability is another desired feature in trajectory studies. One may want to understand the "big picture" of the trajectory as well as details of movements in some particular areas. This means we should be able to view different levels of details when needed, by adjusting the parameters of the algorithm. We will show in the next section how we make our On-Line Data Compression Algorithm for Trajectories, *a.k.a.* OLDCAT be online, adaptive, and scalable.

III. The OLDCAT

As discussed in the previous Section, we assume only minimum information in the raw data: the object ID, time stamp t, and longitude/latitude values (X, Y). For the convenience of discussion, we focus on a single object. The same algorithm and related discussion can easily be extended to the cases of multiple objects.

We assume the raw data stream consists of a series locational data with time stamps, denoted as

$$\mathcal{P} = \left\{ P_{t_0}, P_{t_1}, P_{t_2}, \ldots, P_{t_n}, \ldots \right\}$$

where $P_{t_n} = \left(x, y, t_n \right)$ is referred to as a *data point*. The objective of OLDCAT is to find a sampled subset of $\mathcal{P}$ denoted as $\mathcal{P}' \subset \mathcal{P}$ and to represent $\mathcal{P}$.

$$\mathcal{P}' = \left\{ P_{t_0}, P_{t_i}, P_{t_j}, \ldots \right\}_{0<i<j}$$

We note that the raw data stream may not be continuous, as the signal may disappear when the positioning device is switched off or out of reach. Also, the object may stop moving from time to time. These events break the trajectory into segments. We define 5 different types of points that construct a trajectory:

1) **STAY Point**: where the object stops moving and remain stationary for a period longer than a predefined constant $T_{\max}$. A STAY point itself forms a trajectory segment. Denoted as $\mathbf{S}$.
2) **BEGIN Point**: where a segment of trajectory begins. It marks the location where the signal of the object appears after disappeared for a period of length $T_{\max}$, or the object moves off from a stay point for a distance longer than $D_{\min}$. This minimum distance control parameter $D_{\min}$ is needed to tolerate signal noises and errors. BEGIN points are denoted as $\mathbf{B}$.
3) **END Point**: where a segment of trajectory ends, when signal fades out or object stops moving. A BEGIN point together with the following next END point defines a segment of the trajectory. Denoted as $\mathbf{E}$.
4) **MOVE Point**: where the object moves forward without making significant turns for a distance longer than a predefined constant $D_{\max}$. Denoted as $\mathbf{M}$.
5) **TURN Point**: where the object turns for an angle sharper than a predefined value $\Theta_{\min}$. Denoted as $\mathbf{T}$.

We use $\leftarrow$ to denote the state of P. For example, $P \leftarrow \mathbf{S}$ means P is of type STAY. The parameters ($D_{\max}, D_{\min}, T_{\max}, \Theta_{\min}$) can be adjust to according to the requirement of the trajectory data or the signal quality. This will be discussed in detail in Section IV.

When a new data point P_t is collected, OLDCAT tries to identify whether it belongs to one of the 5 types above. If yes, we will add P_t to $\mathcal{P}'$, otherwise the data point will be disregarded. In the following section, we describe the detection of begin, end and stay points, which is relatively more straight forward than detection of the move and turn points, which will be covered in Section III-B.

A. Identifying BEGIN, END and STAY Points

We create Algorithm 1 for the purpose of finding points of STAY, BEGIN and END type, namely **OLDCAT_SBE**. We use t_c to denote current time, thus P_{t_c} is the latest (current) data point, and $P_{t_{c-1}}$ will be the previous (second latest) data point. $P_{t'}$ is used to denote the last data point added to trajectory $\mathcal{P}'$, thus its time stamp t' will be the largest in $\mathcal{P}'$. We note that before considering adding P_{t_c} to $\mathcal{P}'$ as a BEGIN, END or STAY Point, we should have

1) $P_{t'}P_{t_c} > D_{\min}$: the object has moved off for a minimum distance from the previous data point in the compressed trajectory;
2) $t_c - t_{c-1} > T_{\max}$: the time difference for the last two consecutive data points is larger than the maximum requirement.

Otherwise, P_{t_c} be stored aside in a set $\mathcal{P}_{\text{temp}}$, and handled by procedure **OLDCAT_MT** to check if it should be added as a MOVE or TURN point. This is shown by line 22–23 and 26 in Algorithm 1.

To understand the algorithm, we can see there are four cases where P_{t_c} be added to $\mathcal{P}'$ as BEGIN points:

1) **Case 1**: P_{t_c} is the first data point of the object, as depicted in Algorithm 1 line 2–4;
2) **Case 2**: $P_{t'}$ is a STAY point, and P_{t_c} will be the first data point in the next segment, as depicted in Algorithm 1 line 10–11;
3) **Case 3**: $P_{t'}$ is a BEGIN point. Since
$$t_c - t_{c-1} > t_c - t' > T_{\max} ,$$
$P_{t'}$ needs to be changed as a STAY point, and P_{t_c} will be the BEGIN point in the next segment, as depicted in Algorithm 1 line 13–14;
4) **Case 4**: $P_{t'}$ is a MOVE or TURN point. Since
$$t_c - t_{c-1} > t_c - t' > T_{\max} ,$$
the previous segment (containing $P_{t_{c-1}}$) should have ended. Thus $P_{t_{c-1}}$ will also be added to $\mathcal{P}'$ as the END point of the previous segment, and P_{t_c} will be the BEGIN point in the next segment, as in Algorithm 1 line 17–19;

We note that it is not possible that $P_{t_{c-1}}$ is an END point when the procedure **OLDCAT_SBE** is called. This is because END point will only be created when the BEGIN point of the next trajectory segment is identified, *i.e.* in case 4 discussed above.

Algorithm 1 Identifying **S**, **B**, and **E**
```
 1: procedure OLDCAT_SBE($\mathcal{P}'$, $P_{t_c}$, $P_{t_{c-1}}$)
 2:     if $\mathcal{P}' = \emptyset$ then
 3:         $P_{t_c} \leftarrow$ B                         ▷ First data point
 4:         $\mathcal{P}' = \{P_{t_c}\}$
 5:     else
 6:         $t' = \max\{t | P_t \in \mathcal{P}'\}$
 7:         if $P_{t'}P_{t_c} > D_{\min}$ then
 8:             if $t_c - t_{c-1} > T_{\max}$ then
 9:                 if $P_{t'}$ is S then
10:                     $P_{t_c} \leftarrow$ B        ▷ After a STAY point
11:                     $\mathcal{P}' = \{P_{t_c}\} \cup \mathcal{P}'$
12:                 else if $P_{t'}$ is B then
13:                     $P_{t'} \leftarrow$ S
14:                     $P_{t_c} \leftarrow$ B       ▷ After a BEGIN point
15:                     $\mathcal{P}' = \{P_{t_c}\} \cup \mathcal{P}'$
16:                 else          ▷ After a Move/TURN point
17:                     $P_{t_{c-1}} \leftarrow$ E
18:                     $P_{t_c} \leftarrow$ B
19:                     $\mathcal{P}' = \{P_{t_{c-1}}, P_{t_c}\} \cup \mathcal{P}'$
20:                 end if
21:             else
22:                 $\mathcal{P}_{\text{temp}} = \{P_{t_c}\} \cup \mathcal{P}_{\text{temp}}$
23:                 call OLDCAT_MT($\mathcal{P}'$, $P_{t_c}$, $\mathcal{P}_{\text{temp}}$)
24:             end if
25:         else
26:             $\mathcal{P}_{\text{temp}} = \{P_{t_c}\} \cup \mathcal{P}_{\text{temp}}$
27:         end if
28:     end if
29: end procedure
```

It is easy to see that the running time of **OLDCAT_SBE** is $O(1)$. Since it only takes the last two data points (P_{t_c} and $P_{t_{c-1}}$), it is able to deal with online streaming data with $O(1)$ storage required.

B. Identifying Move and Turn Points

In short, we use "forward looking" to find MOVE points and "backward looking" to identify TURN points once a MOVE point is found. That is, we find a MOVE point first, by measuring the distance between the incoming data point (P_{t_c}) and the last point added to the compressed trajectory ($P_{t'}$). Once a MOVE point is identified, we look back at those data points collected in time interval $[t', t_c]$, which are stored in $\mathcal{P}_{\text{temp}}$ by OLDCAT SBE to find the TURN points where the object makes relatively sharp turns.

1) *Geometry Theorems for Finding TURN Points*: Before we show the algorithm, some geometric theorems need to be introduced.

Theorem 1. (Inscribed Angle Theorem) An angle θ inscribed in a circle is half of the central angle 2θ that subtends the same arc on the circle. Therefore, the angle does not change as its apex is moved to different positions on the circle.

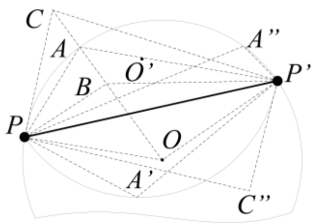


Fig. 1. Inscribed angle.

Proof to Theorem 1 can be found in [13]. In Fig. 1, P and P'' are points on a circle centered at O. According to this Theorem, we can see $\angle PA''P' = \angle PAP'$. Moreover, the following Corollary could be derived:

Corollary 1. (Inscribed Angle Corollary) For the same arc, if the apex is located inside the given circle, the angle will be larger than the inscribed angle; otherwise if it is located outside, it will be smaller than the inscribed angle.

Proof: Let's consider inscribed angle $\angle PAP'$ in Fig. 1. Point B and C are on the line segment of OA and its extension, respectively. In $\triangle PAB$, its exterior angle

$$\angle OBP = \angle BAP + \angle APB > \angle PAB.$$

Similarly, in $\triangle P'AB$, we have $\angle OBP' > \angle P'AB$. Thus

$$\begin{aligned}\angle PAP' &= \angle PAB + \angle BAP' \\ &< \angle PBO + \angle OBP' = \angle PBP'\end{aligned}.$$

According to Theorem 1, this shows that $\angle P'BP$ is larger than any inscribed angle that subtends the same arc with $\angle PAP'$. Using the same principle, we can show that

$$\begin{aligned}\angle PCP' &= \angle PCA + \angle ACP' \\ &< \angle PAB + \angle ABP' = \angle PAP'\end{aligned}.$$

Thus for any given apex located inside (*e.g.* B) or outside *e.g.* C) the circle, we can always locate the corresponding inscribed angle (*e.g.* $\angle PAP'$) by connecting the circle center with the point and extend, and find that Corollary 1 is true.

We note it due to symmetry, on the other side of line segment PP' we can find another circle center other than O (denoted as O'). Its corresponding arc is shown in Fig. 1 as $\widehat{PA'P}$, and Theorem 1 and Corollary 1 also hold to show $\angle PA'P' > \angle PC''P'$.

Thus we are able to draw the conclusion that for any point namely B located inside the "olive" shape $\widehat{PA'P'}\widehat{A''A}$ (which is in fact an *equal-circle-intersection area*), $\angle PBP' > \angle PAP'$; for point C outside this area, $\angle PCP' < \angle PAP'$. Moreover, using analytical geometry techniques, given coordinates of P and P', and the size of $\angle PAP'$, it is not difficult to obtain the coordinates of O and thus determine whether or not a given point is inside the olive area or outside. The calculation steps are intuitive and thus omitted in this paper. We define this procedure as an **OLIVE.CHK**, which could be used in the **OLDCAT_MT** algorithm to find the TURN points.

2) ***OLDCAT_MT***: Let's consider P and P' in Fig. 1 as two different data points in the raw trajectory data departed by a certain amount of time, and thus there could be a series of other data points taken between them. Theorem 1 and Corollary 1 can be used to identify the points where the object makes significant turns (turns with angle sharper than $\Theta_{\min}$, such as at point C or C''), and add them as TURN point to the compressed trajectory $\mathcal{P}'$.

Algorithm 2 Identifying M and T
```
1:  procedure OLDCAT_MT($\mathcal{P}', P_{t_c}, \mathcal{P}_{\text{temp}}$)
2:      $t' = \max\{t | P_t \in \mathcal{P}'\}$
3:      if $P_{t'}P_{t_c} > D_{\max}$ then
4:          $P_{t_c} \leftarrow$ M                          ▷ Add MOVE point
5:          $\mathcal{P}' = \{P_{t_c}\} \cup \mathcal{P}'$
6:          while $\mathcal{P}_{\text{temp}} \neq \emptyset$ do
7:              $t_A = \min\{t | P_t \in \mathcal{P}_{\text{temp}}\}$
8:              $\mathcal{P}_{\text{temp}} = \mathcal{P}_{\text{temp}} \setminus P_{t_A}$
9:              if  OLIVE.CHK($P_{t'}, P_{t_c}, P_{t_A}, \Theta_{\min}$)  &
                    $P_{t'}P_{t_A}, P_{t_A}P_{t_c} > D_{\min}$ then
10:                 $P_{t_A} \leftarrow$ T                  ▷ Add TURN point
11:                 $\mathcal{P}' = \{P_{t_A}\} \cup \mathcal{P}'$
12:                 if $t_A > t'$ then
13:                     $P_{t'} = P_{t_A}$
14:                 end if
15:             end if
16:         end while
17:     else
18:         $\mathcal{P}_{\text{temp}} = \{P_{t_c}\} \cup \mathcal{P}_{\text{temp}}$
19:     end if
20: end procedure

21: procedure OLIVE.CHK($P, P', A, \Theta_{\min}$)
22:     Compute coordinates of $O$ and $O'$
23:     if $AO < OP$ or $AO < O'P'$ then
24:         return FALSE
25:     else
26:         return TRUE
27:     end if
28: end procedure
```

In Algorithm 2, once we find a new data point (P_{t_c}) which is at least $D_{\max}$ away from the last data point in the trajectory ($P_{t'}$), we add it to the compressed trajectory ($\mathcal{P}'$) as a MOVE point, otherwise add to the temporary storage ($\mathcal{P}_{\text{temp}}$) as a future candidate of TURN point.

After adding each MOVE point, the while loop from line 6 to line 16 looks at every data point stored in $\mathcal{P}_{\text{temp}}$ and tries to identify TURN points. In every look the data point with earliest time stamp in $\mathcal{P}_{\text{temp}}$ is taken out of the storage as P_{t_A}. It needs three criteria to qualify as a TURN point:

1) It is at least $D_{\min}$ away from $P_{t'}$;
2) It is at least $D_{\min}$ away from P_{t_c};
3) It is located outside the "olive area" defined by $P_{t'}$, P_{t_c},

and $\Theta_{\min}$.

Again, the use of $D_{\min}$ is to tolerate signal noises and errors.

As discussed in the previous section, an independent procedure **OLIVE.CHK** uses analytical geometry techniques to confirm whether or not P_{t_A} in outside the olive area, *i.e.* represent a turning angle sharper than $\Theta_{\min}$. The outline of the procedure is presented in lie 21–28 of Algorithm 2.

When we add P_{t_A} as a TURN point in the compressed trajectory $\mathcal{P}'$, it is possible that $t_A > t_0$ and P_{t_A} should replace $P_{t'}$ as the last data point in the trajectory before P_{t_c}, as done by line 12–14 in Algorithm 2. This is important because for next data points from $\mathcal{P}_{\text{temp}}$, the new $P_{t'}$ will be used on line 9 for the distance and olive check.

We note that with every MOVE point added, data points storage $\mathcal{P}_{\text{temp}}$ will be removed one by one (line 8). At the end of the while loop, $\mathcal{P}_{\text{temp}}$ should become empty. New data points will be added to $\mathcal{P}_{\text{temp}}$ with new **OLDCAT_SBE** and **OLDCAT_MT** procedures. Thus we could consider $\mathcal{P}_{\text{temp}}$ as a "buffer" with selected items from the incoming data stream. It is cleared once a MOVE point is added, and will not store the entire history of the collected data points.

The running time complexity of **OLIVE.CHK** is $O(1)$. For **OLDCAT_MT** it is $O(n)$, where n is number of data points in Ptemp. It also needs $O(n)$ to store $\mathcal{P}_{\text{temp}}$. Similar to **OLDCAT_SBE**, **OLDCAT_MT** does not require all the historic data points to be stored, and is clearly an online algorithm.

IV. Performance Analysis

OLDCAT offers a compression for the trajectory data with quality that could be quantified. We show how the error margin could be estimated in Sect. IV-A. Moreover, OLDCAT is also flexible in adjusting to different application requirements and signal conditions, which will be demonstrated in Sect. IV-B.

A. Accuracy and Error Tolerance

The accuracy of a trajectory data compression algorithm is determined by the amount of information lost during the compression/sampling process. *Blind spots* are defined as the areas in which the location or direction change of the object may not be detected by the algorithm, and the outcome trajectory $\mathcal{P}'$ may be inaccurate. For OLDCAT, the "Olive Area" between two consecutive data points is its blind spot.

Take Fig. 2 as an example. Let's say P and P' are two data points departed by the distance of $D_{\max}$. The Oliver area could be defined by having $\angle PAP' < \angle PA'P' = \Theta_{\min}$. While data point C — outside the blind spot — could be correctly identified as a TURN point by **OLDCAT_MT** procedure, point D and D' will be disregarded, even though the object is making sharper turns at these point, due to the fact that they are located in the blind spot. Similarly, a lot of details of the curvature trace from P to B and then to P' could be lost since they are in the blind spot. Instead, OLDCAT may compress the trajectory as line segments PB and BP'.

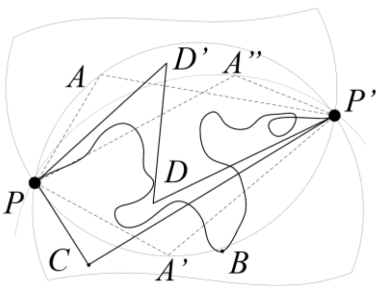


Fig. 2. Blind spot of OLDCAT.

To overcome this, we can reduce $D_{\max}$ or increase $\Theta_{\min}$. For example, if we increase $\Theta_{\min}$ to $\angle PA''P'$, the arc from P to P' will be lower and D' could be identified by **OLDCAT_MT** as a TURN point. This is a trade-off between the details of the trajectory and the compression ratio, which will be discussed in the next section.

With analytical geometry, we could obtain the size of the blind spot (denoted as A_b) as functions of $D_{\max}$ (denoted as d) and $\Theta_{\min}$ (denoted as θ):

$$\mathrm{A}_b = d^2\left(1-\frac{\theta}{\pi}-\sin 2\theta\right)\sec^2\theta$$

The derivation is simple and thus omitted here. We can see from this equation that A_b decreases with smaller d or larger θ, which seals the conclusion we draw earlier.

Another blind spot is from the $D_{\min}$ parameter. In both **OLDCAT_SBE** and **OLDCAT_MT**, we always disregard data points within $D_{\min}$ from an existing point in P'. We do this to tolerate the signal noises and errors.

For example, when the object is stationary, its GPS data could have slight fluctuations. Without the blind area defined by $D_{\min}$, several STAY point might be identified by **OLDCAT_SBE** instead of one. This could add necessary points to the P' and thus reduce the efficiency of OLDCAT. The $D_{\min}$ parameter and its drawback are adjustable. If we are confident that the positioning signal is 100% accurate without any noises or errors, we may set its value to 0 so that the blind spot caused by $D_{\min}$ will be removed. On the other hand, if the signal and the positioning technique are poor, larger $D_{\min}$ should be used to tolerate the signal fluctuations, and thus more details of the trajectory will be lost during compression.

B. Scalability and Sensitivity

In the previous section we have already shown that the performance of OLDCAT is closely related to its parameters ($D_{\max}$, $D_{\min}$, $T_{\max}$, $\Theta_{\min}$). In this section with some example and Fig. 3, we will show in detail how each of them affects the outcome of OLDCAT.

The Raw trajectory data (before compression) is shown in Fig. 3 (a). We add t0 to t6 to the graph to show the direction of the object's movement. Fig. 3b is the reference output of OLDCAT with all the five types of points shown with different notations. A total number of three segments are generated, among which the second segment is of a single STAY point. Fig. 3c – 3f are the outcomes of changing each of the four parameters ($D_{\max}$, $D_{\min}$, $T_{\max}$, $\Theta_{\min}$), respectively. Comparing them to Fig. 3b, we can see the effect of each parameter.

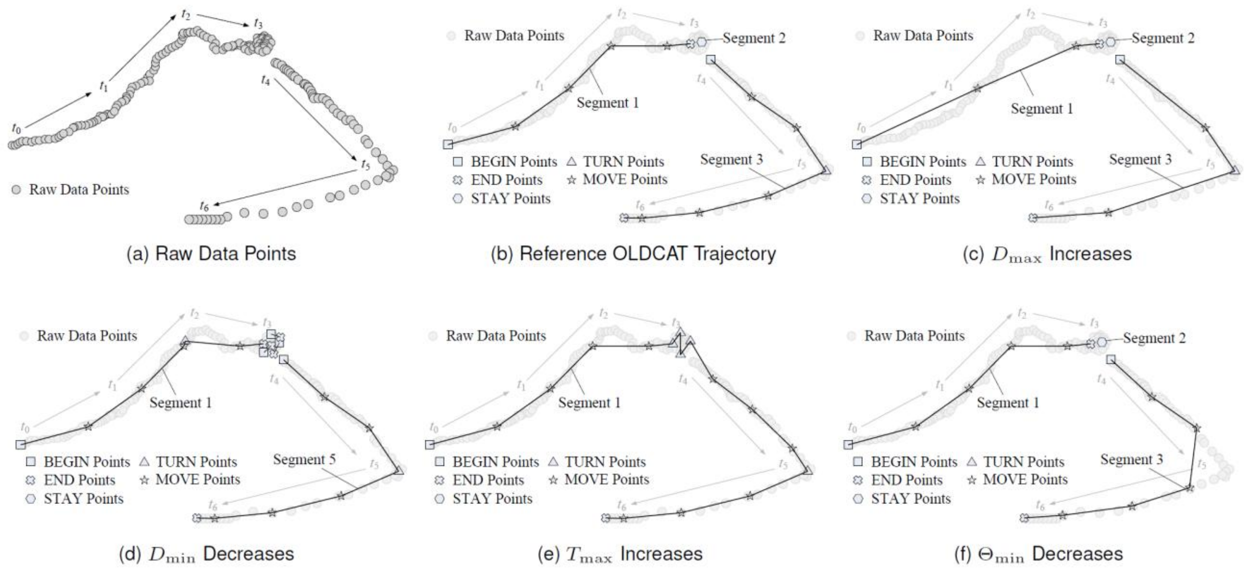


Fig. 3. Changing parameters in OLDCAT.

In Fig. 3c we increase $D_{\max}$. As a result, only points further away will be added by **OLDCAT_MT** as MOVE point. It works fine when the object is moving on a straight course, such as on from t_4 to t_6 in the figure; but when the object moves on a curvature route, like from t_0 to t_3 , a lot of trajectory details are lost in the compression. Therefore, if we want to retain more details of the trajectory, in particular the curvature, smaller $D_{\max}$ should be used.

As we have discussed in the previous section, when we decrease $D_{\min}$, we lower the noise/error tolerance of OLDCAT. We can see this from Fig. 3d. The STAY point is replaced by three additional very short segments generated between t_3 to t_4 by **OLDCAT_SBE**, and an additional TURN point is identified at about t_2 by **OLDCAT_MT**.

The usefulness of these additional data points is arguable. They could be caused by the signal error and noises and thus "fake", but they also could be "true" in the sense that a smaller value of $D_{\min}$ reduces the size of the blind spots, and enables more data points to be added by OLDCAT. No conclusion could be drawn with an actual use case and further knowledge on the positioning signals.

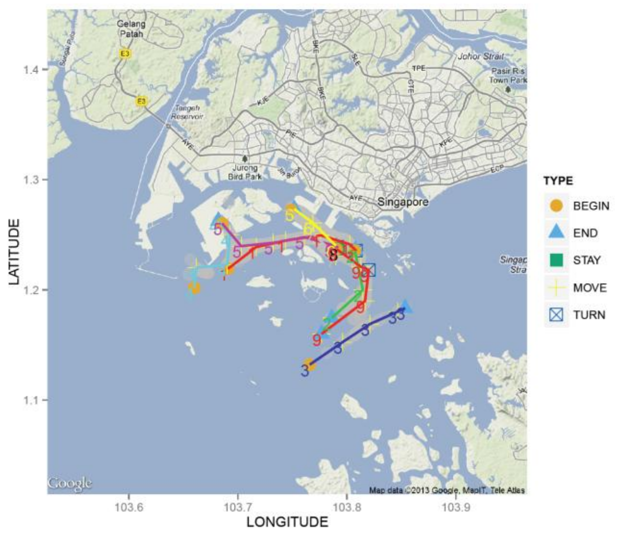
In **OLDCAT_SBE**, the value of $T_{\max}$ determines how we divide the trajectory to different segments. When its value increases, we wait longer for the next data point in the same segment to show up. From Fig. 3e we can see that segment 2 and 3 in the original OLDCAT results are now connected to segment 1. This is because with larger value of $T_{\max}$, the if condition on line 8 of Algorithm 1 becomes more strict, and less points would be considered as BEGIN point. It shows that if we want to study how the trajectory breaks into segments, smaller $T_{\max}$ value should be used.

We have already shown that a smaller $\Theta_{\min}$ value will cause a larger blind spot, and may miss out potential TURN point. This phenomenon is again shown with Fig. 3f. When the value of $\Theta_{\min}$ decreases, the TURN point near t_5 disappears. This makes the compressed trajectory less accurate, but also reduces the number of points in the outcome trajectory and hence improves the compression ratio. Adjusting $\Theta_{\min}$ could be effective when we want to show the trajectory in different scales and detail levels. In particular, if we want to study where and how the object makes turns, different values of $\Theta_{\min}$ should be used to give us a more complete picture.

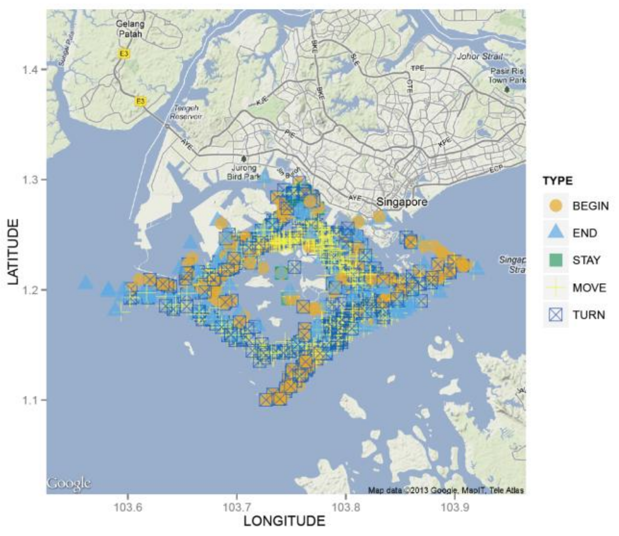
V. Case Study with Harbor Data

Our office is located near the southern shore of Singapore. The Pasir Panjang Container Terminal and Jurong Port lies outside our window. Together, they are one of the busiest ports in the world [14]. We have been collecting the Automatic Identification System (AIS) signals [15] since June 2012. In this study, we are using positioning data of 3000 different ships (randomly sampled out of 3791 in total) over about three weeks' time (from 29 June 2012 to 16 July 2012). The data set (*i.e.* $\mathcal{P}$) has 5708927 lines of records of locations with ship ID and time stamp. Although it is not real time streaming data, we simulated the data stream with stored data when we apply OLDCAT.

After compress the data using OLDCAT, we obtained 92907 points in $\mathcal{P}'$. The compression ratio is 1.63%. In Fig. 4 we show the trajectory obtained by OLDCAT for one single ship and 100 ships respectively. We did not plot the trajectory for all the 3000 ships because the plot is simply overwhelming to make much sense.



a)



b)

Fig. 4. Compressed trajectory OLDCAT.

In Fig. 4 a), we can see 9 segments of the ship's trajectory are identified. In particular, we can see how the ship moves into the port in segment 4, then move out in segment 5. Segment 8 is a single STAY point, and is thus less visible in the figure. This plot shows that with as few as 1% of the data points, we can still re-construct the movement of the object, proving the effectiveness of OLDCAT.

Fig. 4b overlays the trajectories of 100 ships. Although due to the high density of the points in the plot, we may not be able to see the sequence of the points, interestingly, we can learn how the ships behave by studying how different types of the points in $\mathcal{P}'$ distribute over the entire area. For example, a lot of ships make turns at the southern part of the area, indicated by high density of the TURN points (with yellow color), while in the middle they simply keeps moving straight ahead, indicated by the purple color MOVE points. This could be a starting point of studying the behavior and identify regulation violation within the harbor area, which is an interesting extension to the applications of OLDCAT.

VI. Conclusion

In this paper we propose an OnLine Data Compression Algorithm for Trajectory (OLDCAT). It deals with real time positioning data of moving objects, and tries to identify key characteristic points that define the trajectory. By taking these points as samples to represent the trajectory, it compresses the trajectory data to huge extends. The key strengths of OLDCAT include: online computation, very low ($O(n)$) running time and storage space complexity, adjustable scale, and signal error/noise tolerance. We have analytically shown the accuracy of the algorithm, and demonstrated examples and use case studies where OLDCAT can effectively compress trajectory data and identify the movement pattern of the objects.

References

[1] T. L. Nyerges, "Locational referencing and highway segmentation in a geographic information system," *ITE Journal*, vol. 60, no. 3, pp. 27–31, 1990.

[2] N. Andrienko and G. Andrienko, *Exploratory Analysis of Spatial and Temporal Data*, Germany: Springer Berlin, 2006.

[3] I. Forum, *Improving Reliability on Surface Transport Networks*, OECD Publishing, 2010.

[4] G. Wade, *Signal Coding and Processing*, Cambridge University Press, 1994.

[5] Y. Azar, "On-line load balancing," in *Online Algorithms: The State of the Art*, A. Fiat and G. J. Woeginger, Eds., Springer, pp. 178–195, 1998.

[6] R. Bellman, "On the approximation of curves by line segments using dynamic programming," *Commun. ACM*, vol. 4, no. 6, pp. 284, Jun. 1961.

[7] R. E. Bellman, *Dynamic Programming*, Dover Publications, Incorporated, 2003.

[8] J. Kleinberg and E. Tardos, *Algorithm Design*, Pearson Education, 2006.

[9] D. H. Douglas and T. K. Peucker, "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*, vol. 10, no. 2, pp. 112–122, 1973.

[10] N. Meratnia and R. By, "Spatiotemporal compression techniques for moving point objects," *Advances in Database Technology - EDB*T, Heidelberg: Springer Berlin, vol. 2992, pp. 765–782, 2004.

[11] M. Potamias, K. Patroumpas, and T. Sellis, "Sampling trajectory streams with spatiotemporal criteria," in *Proc. 18th International Conference on Scientific and Statistical Database Management*, pp. 275–284, 2006.

[12] F. Schmid, K.-F. Richter, and P. Laube, "Semantic trajectory compression," *Advances in Spatial and Temporal Databases*. Springer, pp. 411–416, 2009.

[13] C. Ogilvy, *Excursions in Geometry*, Dover Publications, Incorporated, 1990.

[14] *World Port Rankings 2012*, American Association of Port Authorities.

[15] IEC Technical Committee 80, “Maritime navigation and radiocommunication equipment and systems,” *International Electrotechnical Commission (IE*C), Tech. Rep., 1980.

Wang Ting was born in 1983 in Chengdu, Sichuan, China. He came to Singapore for his undergraduate studies under Singapore Government-Linked Company SM2 scholarship in 2001. He obtained his Bachelor’s degree with honors in 2005 and subsequently Ph.D in 2011 both at Nanyang Technological University (NTU), Singapore.

He worked as a demand planner at Apple South Asia and joined SAP as a Data Scientist in 2012. His research interests include data mining, mathematical modeling and algorithmic optimization.

Dr. Wang loves soccer. He considers family as his greatest award. He has a son, and he is a good cook — said his wife.