Rights Protection for Trajectory Streams

Mingliang Yue[1], Zhiyong Peng[1], Kai Zheng[2], and Yuwei Peng[1,⋆]

[1] School of Computer, Wuhan University,
Wuhan, Hubei, 430072, China
ywpeng@whu.edu.cn

[2] School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane, Australia
kevinz@itee.uq.edu.au

Abstract. More and more trajectory data are available as streams due to the unprecedented prevalence of mobile positioning devices. Meanwhile, an increasing number of applications are designed to be dependent on real-time trajectory streams. Therefore, the protection of ownership rights over such data becomes a necessity. In this paper, we propose an online watermarking scheme that can be used for the rights protection of trajectory streams. The scheme works in a finite window, single-pass streaming model. It embeds watermark by modifying feature distances extracted from the streams. The fact that these feature distances can be recovered ensures a consistent overlap between the recovered watermark and the embedded one. Experimental results verify the robustness of the scheme against domain-specific attacks, including geometric transformations, noise addition, trajectory segmentation and compression.

Keywords: Rights protection, trajectory streams, watermarking, robustness.

1 Introduction

Recent advances in mobile positioning devices such as smart phones and in-car navigation units made it possible for users to collect large amounts of GPS trajectories. After expensive and laborious data acquiring process, companies and institutions frequently outsource their trajectory data for profit or research collaborations. Therefore, the rights protection for the owner over the precious data becomes an important issue.

Watermarking is one of the most important techniques that can be used for the rights protection of digital data. Basically, watermarking means slightly modifying the host data and forcing it to imply certain secret information. The information (i.e., watermark) is identifiable and can be detected from the (possibly modified) data for ownership assertion [1]. In fact, watermarking is predominantly used for the rights protection of digital contents, e.g., images [2], audios [3], videos [4] and vector maps [5][6][7]. For trajectory databases, two watermarking schemes have been proposed [8][9]. Given a trajectory set, both schemes

⋆ Corresponding Author.

embed watermark by slightly modifying coordinates of locations in trajectories. And both schemes are robust against certain data modifications (attacks) such as geometric transformations and noise addition.

However, as other kinds of data collected from sensors, trajectory data often come to databases in a streaming fashion. As Sion et. al noted [10][11], batched watermarking schemes, such as schemes in [8] and [9], are not applicable to streaming data. Such schemes can embed and detect watermark only when the entire dataset is available, while attacks may come before the entire dataset is collected. For example, suppose taxi company A obtains trajectories from its affiliated taxies and sells the data to service provider B for real-time passenger-hunting recommendation (for taxi drivers) [12]. If the data is not watermarked, and there is another service provider C who needs the data to support the passenger ridesharing service [13], then B can simply re-direct the trajectory stream to C for profit. In this scenario, online watermarking schemes [10][11] should be employed to embed watermark immediately after the taxi company receiving the data.

Moreover, any (batched or online) watermarking scheme should be robust against certain domain-specific transformations (attacks). For trajectory data, attacks like geometric transformations and noise addition have been considered and handled in [8] and [9]. However, in those two schemes, two important data operations are not considered: trajectory segmentation and trajectory compression. Simply speaking, trajectory segmentation is used to filter out subsets of trajectory locations for sub-trajectory mining [14][15], while trajectory compression aims to reduce the size of trajectory data to resolve the inefficiency in data transmission, querying, mining and rendering processes [16][17][18]. Both operations (attacks), may not influence the usability of trajectory data in certain scenarios, however, can modify the data in a significant magnitude, and in turn, harm the watermark embedded in the data. A watermarking scheme designed for trajectory data should have the ability to handle those two attacks.

Considering the problems mentioned, in this paper, we propose for the first time an online watermarking scheme for the rights protection of trajectory streams. The scheme operates in a finite window, single-pass streaming model. The main idea is to embed watermark into the feature distances, i.e., distances between pairs of feature locations in trajectories, and the feature locations are identified using the proposed Time Interpolated Feature Location Selection algorithm. In addition to traditional attacks (i.e., geometric transformations and noise addition), our scheme is also robust against trajectory segmentation and compression. Our contributions include (1) the identification of the problem of watermarking trajectory streams and major types of attacks on the data; (2) the design and analysis of new watermarking scheme for the data; (3) a proof of the scheme's robustness against the considered attacks based on experimental evaluation.

The remainder of this paper is organized as follows. Section 2.4 outlines the major challenges. It introduces the scenario and the underlying processing model, discusses associated attacks and overviews related work. Then, the proposed online watermarking scheme is given in Section 3. Finally, performance study and conclusions are given in Section 4 and 5 respectively.

2 Challenges

2.1 Scenario

Fig. 1 shows the general scenario of watermarking streaming data, which was firstly demonstrated in [10]. In the scenario, streaming data is modeled as an (almost) infinite timed sequence of values of a particular type (e.g., temperature, stock market data). The watermarking technique is used to deter a malicious customer (licensed data user B in Fig. 1), with direct stream access, to re-sell possibly modified trajectory streams to others (unlicensed data user C for example) for profit. The challenges are that the underlying watermarking scheme should operate in a finite window, single pass model (i.e., online model), and needs to be robust against domain-specific attacks.

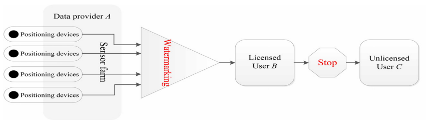

Fig. 1. Stream Watermarking Scenario

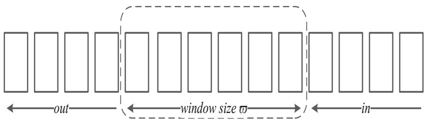

Fig. 2. Space Bounded Processing Window

2.2 Processing Model

The stream processing should be both time and space bounded [10]. The time bound derives from the fact that it has to keep up with incoming data. For space bound, as demonstrated in Fig. 2, we model the space by the concept of a *processing window* of size ϖ: no more than ϖ of locations can be stored locally at the processing time. As more incoming data become available, the scheme should push older locations out to free up space for new locations.

2.3 Attack Model

The attacks that can be applied to the watermarked data are domain-specific. In this paper, since our purpose is to watermark trajectory streams, we identify several meaningful attacks on trajectory data as: ($A1$) geometric transformations (i.e., translation, rotation and scaling), ($A2$) noise addition, ($A3$) trajectory segmentation, and ($A4$) trajectory compression. $A1$ and $A2$ are intuitive [8][9] hence no further explanation will be outlined here. While $A3$ and $A4$ are more complicated, before explaining these two attacks, we give three definitions as follows [23].

A ***Trajectory Stream*** T is a possibly unbounded sequence of time-stamped locations, denoted as $T = \{\langle x_1, y_1, t_1\rangle, \langle x_2, y_2, t_2\rangle, \ldots, \langle x_n, y_n, t_n\rangle, \ldots\}$, where t_i is an element of a numerable, disjoint *Time Schedule* defined as $TS = \{0, 1, 2, \ldots\}$, x_i, y_i represent geographic coordinates of the moving object at sampling time t_i.

The (a, b)***-Segmented Trajectory*** $T_{a:b}$ of a trajectory stream T is a subset of consecutive locations of T, denoted as $T_{a:b} = \{\langle x_i, y_i, t_i\rangle | a \leq t_i \leq b, \langle x_i, y_i, t_i\rangle \in T\}$.

Given a distance threshold ε, trajectory T^ε is a ε***-Simplified Trajectory*** of a trajectory stream T, if $T^\varepsilon \subsetneqq T$, $\forall L_{t_i} = \langle x_i, y_i, t_i\rangle \in T - T^\varepsilon$, $dist(L_{t_i}, T^\varepsilon) \leq \varepsilon$, where $dist()$ is a certain distance function. $\forall L_{t_i} \in T^\varepsilon$, we call it a ***feature location***.

Trajectory segmentation (compression) attack means extracting segmented (simplified) trajectories and re-streaming them for profit. The implementation of trajectory segmentation is straightforward. While for trajectory compression, many online trajectory simplification algorithms can be employed for attack [16][18] [19][20][21]. Those algorithms also take compression ratio (i.e., the size of the original trajectory divided by the size of the compressed trajectory) as threshold, and return simplified trajectories satisfying compression ratio constraints as compressed trajectories. Both attacks, may not influence the usability of trajectory data in certain scenarios [14][15][19][20], however, can modify the data in a significant magnitude, and in turn, harm the watermark embedded in the data. In this paper, all the attacks ($A1$ to $A4$) will be properly considered and handled. To the authors' knowledge, this is the *first piece of rights protection work* that takes trajectory segmentation and compression into account as attacks.

2.4 Related Work

An Online Watermarking Scheme (OLWS) has been proposed in [11] (which is extended based on the work in [10]) to watermark streaming data. The scheme takes general data streams (e.g., temperature, stock market data, etc.) as input. The attacks considered are summarization, extreme sampling, segmentation, geometric transformations and noise addition. During the embedding, the scheme firstly initializes a data normalization process, and then identifies major extremes (which are composed of values at and close to local minimums or maximums) based on the normalized data. Finally, the scheme embeds one watermark bit into all the values in a major extreme if the extreme satisfies a certain selection criterion.

The scheme works well for general data streams, however, it is not suitable for trajectory streams. On the one hand, the attacks faced by an OLWS for trajectory streams are different (see Section 2.3). For example, the scheme may be ineffective under trajectory compression: locations with local minimum or maximum coordinates are not necessarily the feature locations in a trajectory, and may be deleted after compression. On the other hand, the robustness against geometric transformations of the scheme mainly depends on the data normalization process, which can only be done based on a known data distribution. If the distribution is unknown, an additional initial *discovery* run should be employed to learn one. For trajectories that are composed of locations representing the motion of real objects, such distribution may not exist or is difficult to find out. Therefore, the technique of this work cannot be employed in this paper to handle geometric transformation attacks for trajectory streams.

The resistance to geometric transformations and noise addition have been achieved in two batched trajectory watermarking schemes [8][9]. The scheme in [8] and [9] embeds watermark into distance ratios and Fourier coefficients of locations respectively. Since both the cover data are geometrical invariants, the schemes can properly withstand geometric transformations. And the resistance of the schemes to noise addition is achieved by embedding the watermark bits into the properly selected bits of the cover data. These two schemes also have two problems for trajectory streams. First, none of the schemes operates in online setting, which makes the schemes not applicable for the scenario. Second, both schemes embed watermark based on all the locations in the trajectories. Hence, they are fragile to trajectory compression attack (work in [9] is also fragile to trajectory segmentation).

In summary, due to the unique properties of streaming trajectory data and the corresponding attacks, an online scheme for watermarking trajectory streams that can handle the attacks outlined in Section 2.3 is in need.

3 Watermarking Trajectory Streams

The basic idea of the proposed watermarking scheme is based on two observations. First, feature locations of trajectories will survive trajectory compression attack, if the malicious attacker wants to preserve the usability of the trajectories. Second, distances between pairs of locations will not change if the trajectory is translated or rotated as a whole.

Therefore, the proposed watermarking scheme embeds watermark into feature distances (i.e., distances between pairs of feature locations) with the following process: (1) identify the feature locations in the trajectory stream as processing window advancing; (2) if two consecutive feature locations appear in a same processing window, calculate the (feature) distance between the locations; (3) determine a watermark bit of the global watermark based on the distance according to a selection criterion, and embed the bit into the distance. The fact that these feature distances can be recovered ensures a consistent overlap (or even complete identity) between the recovered watermark and the embedded one (in the un-attacked data). In the watermark detection process, (4) the feature locations are identified and the feature distances are calculated; for every feature distance, (5) the selection criterion in (3) is used once again to match the distance and the watermark bit, the corresponding 1-bit watermark is extracted, and ultimately the global watermark is gradually reconstructed, by using majority voting.

In the following we firstly introduce the proposed *Time Interpolated Feature Location Selection* (TIFLS) algorithm in Section 3.1. The algorithm can identify stable feature locations from an attacked trajectory stream. Then, we give the watermark embedding method in Section 3.2, by employing TIFLS for feature location identification. The watermark detection method is introduced in Section 3.3. At last, parameters used in the scheme are discussed in Section 3.4.

3.1 Time Interpolated Feature Location Selection

To guarantee that the embedded watermark is still detectable after various attacks, the feature locations used for watermark embedding should be recovered after attack. Many online simplification algorithms have been proposed to identify feature locations in a trajectory stream [16][18][19][20][21]. The basic intuition behind those algorithms is that the locally characteristic locations may also be characteristic in a global view. Hence, the process of feature location selection can be described as, (1) fill up the processing window with incoming locations; (2) select feature locations in the current window based on a certain batched simplification method (e.g., Douglas-Peucher algorithm in [22]), free the window and repeat the process. The intuition is reasonable and the results are satisfactory. However, these solutions for feature location selection are ineffective in our scenario: for a possibly compressed and/or segmented trajectory, the selected feature locations may change (i.e., cannot be recovered), since the locations filled in every processing window may be very different from those fetched from the original trajectory.

Therefore, in this paper, we identify feature locations based on *Time Window* defined as follows. A ***Time Window*** TW_m is a ω-sized consecutive subset of TS, denoted as $TW_m = \{t_s, \ldots, t_e\}$, where $m \in \{0, 1, 2, \ldots\}$, $t_s, t_e \in TS$, $t_s = \omega * m$, $t_e = t_s + \omega - 1$. Then, given a trajectory stream T, and a time window TW_m, we can deduce a projection of T on TW_m as $T_m = \{< x_i, y_i, t_i > | t_i \in TW_m, < x_i, y_i, t_i > \in T\}$

Based on the notations, given a threshold ϵ, for every TW_m, our *Time Interpolated Feature Location Selection* (TIFLS) algorithm identifies a location as feature location from its corresponding T_m if (1) among other locations in T_m, the location has the largest Synchronous Euclidean Distance (SED) according to the reference line; (2) its SED is larger than ϵ. The reference line is the line connecting the locations with boundary times of TW_m as sampling times (i.e., L_{t_s} and L_{t_e}, we call these locations boundary locations hereafter).

TIFLS is not a simplification algorithm. Rather, it identifies the location which meets the characteristic conditions in each T_m as feature location for watermark embedding. Hence, these feature locations are very likely to be recovered during watermark detection, even after attack. One can deduce that if a segmented trajectory covers more than two time windows, then in the worst case, trajectory segmentation only influences the first and the last T_m with respect to the segmented trajectory stream. The feature location in each *inner* T_m can be properly recovered. Meanwhile, after trajectory compression (denote T^c the compressed trajectory stream), every T^c_m is a subset of the original T_m. Compared with other locations, the feature locations will more likely get through trajectory compression, and be identified by TIFLS.

The only problem left is that the boundary locations may not exist in the input trajectory. We solve the problem by interpolating a virtual location for every missing boundary location, based on the locations that exist in the trajectory and adjacent to the boundary location. Given two locations L_a and L_c, $a < c-1$, interpolating a location L_b ($a < b < c$) means finding the L_b with $\overrightarrow{v}_{ab} = \frac{b-a}{c-a}\overrightarrow{v}_{ac}$, where $\overrightarrow{v}_{ij}$ denote the vector from L_i to L_j in the Euclidean plane. The virtual boundary location can excellently simulate the real location, since the SED between any boundary location deleted due to compression and the compressed trajectory should be less than a certain threshold (see the notation of compression attack in Section 2.3).

3.2 Watermark Embedding

To synchronize processing window and time window, we define the time coverage of a processing window as follows. The ***Time Coverage*** TC of a processing window is a consecutive subset of TS, denoted as $TC = \{t_f, \ldots, t_l\}$, where t_f and t_l is the sampling time of the first and the last location in the processing window respectively.

Let TC_c denote the time coverage of the *current* processing window, our watermark embedding algorithm can be described as follows. In the **first step**, (1) forward the processing window, apply TIFLS on every T_m with $TW_m \subsetneqq TC_c$, until a feature location L_f is identified; (2) push out the locations previous to L_f, i.e., the first location in the current processing window is L_f, a feature location.

Then, in the **second step**, as locations streaming in, as long as a time window TW_m is completely covered by TC_c (i.e., $TW_m \subsetneqq TC_c$), apply TIFLS on its corresponding T_m for feature location identification. If a feature location is identified, go to the third step. Otherwise, if no feature location is identified until the processing window is full, push out the locations in and previous to the last T_m with $TW_m \subsetneqq TC_c$, and return to the first step.

Let W be the watermark to be embedded, each $w_i \in \{0, 1\}$ be the i-th bit of W, and L_s be the feature location identified in the second step. In the **third step**, we embed one watermark bit into L_s as follows. (1) calculate the distance d_{fs} between L_f and L_s as $d_{fs} = ||\overrightarrow{v}_{fs}||$, where $|| \circ ||$ signifies the L_2 norm of a vector. Let β control the bitwise position in d_{fs} in which the watermark bit will be embedded, and $msb(d, \beta)$ return the bits of d that are higher than β. (2) calculate i as $i = H(msb(d_{fs}, \beta), k)\ mod\ l$, where $H()$ is a cryptographic hash function such as MD5, l is the bit length of W, and k is a secret key given by data owner. (3) replace the β-th bit of d_{fs} as the ith bit of W, w_i, to get the watermarked distance d^w_{fs}. (4) find the L^w_s satisfying $\overrightarrow{v}^w_{fs} = \frac{d^w_{fs}}{d_{fs}}\overrightarrow{v}_{fs}$, where L^w_s is exactly the watermarked location of L_s, and $\overrightarrow{v}^w_{fs}$ is the vector from L_f to L^w_s in in the Euclidean plane. **Finally**, push out the locations previous to L^w_s from the current processing window, denote L^w_s as the current L_f, return to the second step.

To guarantee at least two complete time windows can be synchronized in a same processing window, ω should be set to be less than or equal to $\varpi/3$. The

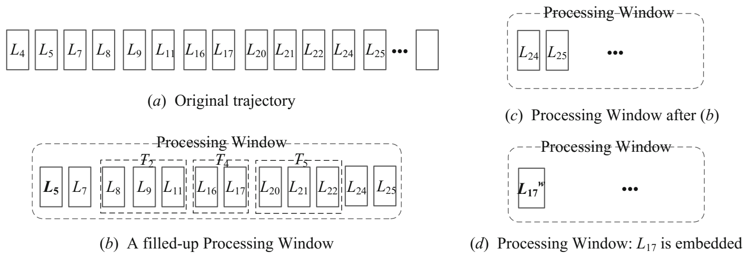
L4 L5 L7 L8 L9 L11 L16 L17 L20 L21 L22 L24 L25 •••
(a) Original trajectory
Processing Window
L24 L25 •••
(c) Processing Window after (b)
Processing Window
T2 T4 T5
L5 L7 L8 L9 L11 L16 L17 L20 L21 L22 L24 L25
(b) A filled-up Processing Window
Processing Window
L17'' •••
(d) Processing Window: L17 is embedded

Fig. 3. Watermark Embedding Process

cryptographic hash function is employed in (2) of the third step to force the malicious attacker into a guessing with respect to the watermark bit embedded. Its power derives strength from both the one-wayness and randomness properties [24]. While the reason behind the use of the most significant bits of d_{fs} is resilience to minor alterations and errors due to watermark embedding and noise addition attack. The embedding strategy employed in (3) of the third step can be easily extended to use quantization index modulation [25] (as we did in the experiments). The method can provide the watermarking scheme with resistance to noise distortion. One can refer to [25] for details about the method.

Fig. 3 illustrates the watermark embedding process, in which ϖ is set to 12, ω to 4. Fig. 3 (a) shows the original trajectory. Fig. 3 (b) demonstrates the filled-up processing window, by assuming the feature location identified in the first step is L_5, and no feature location is identified in the second step. The time coverage of this processing window is $TC_c = \{5, 6, \ldots, 25\}$. Hence TW_2, TW_3, TW_4 and TW_5 are completely covered by TC_c, and T_2, T_4 and T_5 has been considered by TILFS for feature location identification. Fig. 3 (c) demonstrates the next processing window corresponding to Fig. 3 (b): locations previous to L_{24} are pushed out. Fig. 3 (d) demonstrates the processing window after a feature location (L_{17} in this particular example) is identified and a watermark bit is embedded.

3.3 Watermark Detection

In the detection process, the watermark is gradually reconstructed as more and more locations are processed. The reconstruction relies on an array of majority voting buckets as follows. For each bit w_i in the original watermark W, let B_i^0 and B_i^1 be the buckets (unsigned integers) which are incremented each time a corresponding 0/1 bit w_i^d is recovered from the streams. The actual w_i will be estimated by the difference between B_i^0 and B_i^1, i.e., if $B_i^0 - B_i^1 > \vartheta$, then the estimated value for this particular bit becomes $w_i^e = 0$, otherwise, if $B_i^1 - B_i^0 > \vartheta$, $w_i^e = 1$, where $\vartheta \geq 0$. The rationale behind this mechanism is that for un-watermarked streams, the probability of detecting $w_i^d = 0$ and $w_i^d = 1$ would be equal, thus yielding virtually identical values for B_i^0 and B_i^1. In this case, w_i will be undefined and the object will be regarded as un-watermarked [1].

Detection starts by identifying feature locations and calculating feature distances. As long as a feature distance d_i is calculated, the corresponding $j = H(msb(d_i, \beta), k)\ mod\ l$ is determined, then d_i was likely used for the embedding of the j-th bit of W. The bit is then extracted from the β-th bit of d_i depending on its value. Then, the corresponding bucket B_j^0 or B_j^1 is incremented by 1.

3.4 Discussion on Parameters

Except watermark W and secret key k, four parameters should be determined and used in both watermark embedding and detection, i.e., ϖ, ω, ϵ and β. W and k are common to all watermarking schemes. They have been discussed in-depth in many previous works [3, 6, 16]. Therefore, we focus on the latter four parameters. The size of the processing window, ϖ, should be large enough so that it is possible to identify at least two feature locations in most processing windows. ω and ϵ control how characteristic the identified feature locations can be. More characteristic locations have greater chance to be preserved after compression. We will discuss the selection of ω and ϵ in Section 4.1. Consequently, ϖ can be determined based on the ω and ϵ used. As to β, it controls the trade-off between the error introduced and the robustness against noise addition. Obviously, modification on higher bits of distances will result in larger errors on the trajectory data. On the other hand, watermark embedded in the lower bits tends to be more fragile to noise addition. Same as [8], we set β around the bit position of the data's tolerance precision to ensure data fidelity.

4 Experimental Results

In this section, we give an empirical study of the proposed watermarking scheme. The watermarking algorithm was implemented in C++ and run on a computer with Intel Core CPU (1.8GHz) and 512M RAM. The dataset used in the evaluation is obtained from the T-Drive trajectory data sample provided by Microsoft T-Drive project [26][27]. The dataset contains a one-week trajectories of 10,357 taxis. The total number of locations in this dataset is about 15 million and the total distance of the trajectories reaches 9 million kilometers. Each file of this dataset, which is named by the taxi ID, contains the trajectories of one taxi.

Due to the nature of the data, in the experiments, we simulated for each taxi, a trajectory stream with one location per 5-second as sampling rate (incoming data per time unit). During the embedding and detection, β was set as -5 since the precision tolerance of the location's coordinates is $\tau = 10^{-5}$. In the following we verify in turn (1) the performance of TIFLS under various ω and ϵ; (2) the robustness of our method against various attacks; (3) the time overhead and the impact on data quality of the scheme.

[1] Considering also the associated random walk probability [11], we set ϑ as $\sqrt{\frac{B_i^0+B_i^1}{\pi}}$.

4.1 Evaluation on ω and ϵ

As we have stated in Section 3.4, the feature locations identified for watermark embedding should be characteristic enough to withstand trajectory compression. However, how characteristic the feature locations should be is highly related to the compression ratio that the scheme aims to resist. As verified in [20], existing trajectory compression algorithms can compress trajectory data in a compression ratio up to 10 without resulting in significant SED error on the data. The compression ratio is defined as the size of the original trajectory divided by the size of the compressed trajectory [20]. That is, to ensure the usability of the proposed scheme in real applications, (1) 10% (or less) locations should be identified as feature locations for embedding, (2) the feature locations identified should be actually (or positively) characteristic to withstand compression.

In the experiment, we model the two requirements by the concepts of Selection Ratio (SR) and Characteristic Ratio (CR). SR represents the ratio of the number of feature locations identified by TIFLS (denoted as f) with respect to the total number of locations in the trajectories (denoted as n). CR represents the ratio of the number of positive feature locations (denoted as f_p) with respect to the total number of feature locations identified by TIFLS. While by setting the compression ratio of Douglas-Peucker algorithm to 10, the positive feature locations are the locations reported by both TIFLS and Douglas-Peucker algorithm. Namely, we have $SR = f/n$ and $CR = f_p/f$. We use the feature locations selected by Douglas-Peucker algorithm as reference to evaluate the precision of TIFLS since as an optimization algorithm, the performance of Douglas-Peucker algorithm is widely acknowledged. And the algorithm has been extensively employed in almost all the trajectory compression work to evaluate the performance of their methods [16][18][19][20][21].

We applied TIFLS on 1000 trajectories (each containing 10000 locations) for feature location identification, and recorded the SR and CR with respect to different pairs of ω and ϵ. The results are presented in Table 1. From the results, the following conclusions can be drawn: (1) the size of time window ω can be employed to control the SR of the identified locations. Since given a certain ω, the upper bound of SR is $1/\omega$. (2) in general, the lower the SR is, the higher CR will be. However, CR is primarily determined by ϵ. A larger ϵ can lead to a higher CR. If the ϵ is big enough, even a CR of 1 can be achieved. In real world applications, ω can be set as the compression ratio the scheme needs to resist. For ϵ, an initial determination process can be carried out on a small data sample (a one-day trajectory streams for example) to ensure a high CR (e.g., CR$\geq$ 0.9).

4.2 Evaluation on Robustness

In the following experiments, watermarks were first embedded into the trajectory streams. Then, taking the attacked trajectory streams as input, the watermarks were reconstructed using the detection method. For a watermarking scheme, robustness means the scheme can detect a reasonable amount of watermark bits

Table 1. Evaluation on Different ω and ϵ

ω	ϵ	SR	CR
	0	0.1	0.481
	0.00005	0.069	0.866
10	0.0001	0.061	0.894
	0.0005	0.042	0.957
	0.01	0.001	1
	0	0.05	0.568
	0.00005	0.045	0.863
20	0.0001	0.039	0.906
	0.0005	0.023	0.951
	0.01	0.001	1

from watermarked data after attacks. Hence, after watermark extraction, the match rate (MR) is calculated to assess robustness of the scheme, where MR is defined as the ratio of the number of identical bits between the extracted and the embedded watermark to the watermark length. Generally, a dataset can be regarded as watermarked if MR is larger than a given threshold ζ [28]. One can refer to [28] for the selection of ζ. In the following demonstration, the ω and ϵ used in the experiments was 10 and 0.0001 respectively. The size of processing window, ϖ, was set as 30 since averagely a feature location can be reported from every 15 locations (since the CR is 0.061).

In the experiments, one important fact that should be noted is that due to the infinite nature of stream data, watermark detection is a continuous process. It takes time for the watermark detection to be convergent [10]. Our experimental results show that the detection converges when averagely each bit receives 60 votes [2]. In the following experiments, we always record the converged MR under various attacks. For a certain attack, the final MR is the average of the converged MRs reported in different processes using randomly generated watermarks with 64, 128 and 256 as watermark length.

4.2.1 Geometrical Attacks

In this experiment, we applied translation, rotation and scaling to the watermarked trajectory streams. The magnitudes of the attacks are measured with relative coordinate offset, rotation angle and scaling factor, and were set to [-100%, 100%], [-180°, 180°] and [0.1, 2]. The experimental results show that if the trajectories are only translated and rotated (i.e., Scaling = 1), the MRs are the same and equal to 1. For scaled trajectories, the scheme needs to transform the coordinates back into their values on the original scale before feature locations identification and feature distance calculation. This re-scaling and detection has been employed in various batched watermarking schemes [6][7]. According to our experimental results, the method is feasible, and the MRs after re-scaling are equal to 1. The experiment verifies the good preference of the proposed scheme under geometrical attacks.

[2] This means averagely 2000 locations can support a convergent detection of 1-bit, hence a total of 7500 bits can be embedded into the whole dataset.

4.2.2 Noise Addition

To verify the resilience of the proposed scheme to noise addition, we insert random noise to the locations' coordinates in the watermarked trajectory streams. The assumption is that the introduction of noise should not degrade data usability. Hence, the noise added is assumed to be uniformly distributed in the $[0, \chi]$ interval, where $\chi \leq \tau$. The experimental results with respect to different value of χ are presented in Fig. 4. The figure demonstrates that (1) if the attack magnitude χ is less than 0.2τ, the MR remains as 1. This is guaranteed by the power of quantization index modulation. (2) MR decreases with an increasing χ when χ exceeds 0.2τ. However, even when the χ reaches τ, the watermark can still be recovered with a considerably high MR.

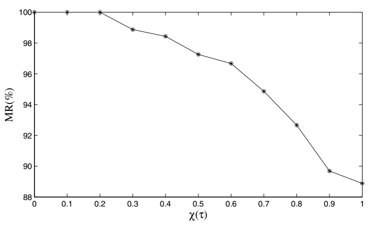

Fig. 4. Resistance to Noise Addition

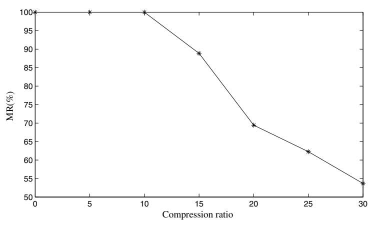

Fig. 5. Resistance to Compression

4.2.3 Segmentation and Compression

Due to the processing model of our scheme, an *active segment* that contains more than ϖ locations or covers 2ω or larger time interval is detectable. Hence, our scheme can naturally resist trajectory segmentation. The conclusion was also verified by our experiments: with sufficient active segments, all the watermark bits can be properly recovered. Namely, as long as the active segments are sufficient enough to support an convergent detection, a MR of 1 can always be achieved.

As to trajectory compression, we compressed the watermarked trajectory streams using Douglas-Peucker algorithm, by setting compression ratio as 0, 5, 10, 15, 20, 25 and 30 respectively. The detection results corresponding to various compressed sets are demonstrated in Fig. 5. As we can see, (1) the entire watermark can be properly recovered, when the compression ratio is less than 10. The reason is that the majority of feature locations survived from the compression attack and can be recovered during the detection. The majority voting ensures a consistent overlap between the detected watermark and the embedded one. (2) MR decreases to 0.88 (a considerably high level) when compression ratio reaches 15, since after compression, only 6.67% locations are left in the compressed trajectories. These locations are not very consistent with the feature locations reported by TIFLS, which makes a small portion of the feature locations used during the embedding unrecoverable in the detection. (3) MR decreases dramatically when compression ratio reaches 20. The reason is intuitive: larger compression ratio causes fewer recovered feature locations. When compression ratio is 30, only 3.33% locations are left for detection. Even all these locations were used for embedding and recovered for detection, the feature distances may change since the consecutive relations among feature locations change. In this case, the parameters used for the feature location identification (i.e., ω and ϵ) should be enlarged to select even more characteristic locations for watermark embedding.

Another more implicit fact to be noted is that boundary location interpolation in TIFLS may also cause the selection of wrong feature locations. However, in the perspective of watermark detection, this is only an extra 'bad' consequence caused by trajectory compression. The good performance of TIFLS for selecting stable feature locations has been implied by the perfect MR facing reasonable compression attacks.

4.3 Overhead and Impact on Data Quality

We performed experiments aimed at evaluating the computation overhead of the proposed watermarking scheme. By far the most computationally intensive operation is the feature location selection. Fortunately, the selection is a single pass process, which means the time complexity of the scheme is $O(n)$. In this experiment, to avoid the time consumed by waiting for the sampling of new locations, the sampling rate was set to infinity: during the processing, all the locations are assumed available as soon as they need to be pushed into the processing window. We tested the embedding and detection on 100 trajectories, each has 10000 locations. The process cost $3340ms$, which means averagely the processing of each processing window needs approximately $50\mu s$ (67000 processing windows have been processed in the embedding and detection). The time consumed is negligible with respective to the time waiting for a processing window to be filled up ($5s * 30 = 150s$). Note that our scheme will introduce a maximum transmission delay of $5 * \varpi$ seconds due to the watermark embedding. However, according to our experimental results, the average transmission delay of the scheme is only $2.5 * \varpi$. According to the User Guide of T-Drive Data [27], this delay only introduces an average error of 282 meters for the stream receiver (when $\varpi = 30$). The error can be handled by many trajectory prediction methods introduced in [29].

We also performed experiments evaluating the impact of our watermark embedding on data quality. We adopt the relative error ξ defined in [9]: $\xi = \sum\limits_{T} \frac{||T-T^w||}{||T||}$, where T and T^w represents the original and the corresponding watermarked trajectory respectively. Our experimental results show that the average value of ξ over a large number (1000+) of runs is 0.23%, much less than 1%, the bound that can lead to a visible error [9].

5 Conclusions

In this paper, we investigated the problem of rights protection for trajectory streams. We proposed an online watermarking scheme that is resilient to various common trajectory transformations. We implemented the proposed watermarking algorithm and evaluated it experimentally on real data. The method proves to be resilient to all the considered transformations, including geometric transformations, noise addition, trajectory segmentation and compression. For future work we plan to consider the online content authentication for trajectory streams, which is another important issue related to streaming data security.

Acknowledgments. This work is supported by the National Natural Science Foundations of China under grant (No.61100019 and No. 61262021). Our appreciation also goes to Liao Zhang for his contribution in the experiment of this paper.

References

1. Petitcolas, F.A.P., Anderson, R.J., Kuhn, M.G.: Information Hiding: A survey. Proceedings of the IEEE 87(7), 1062–1078 (1999)
2. Wang, Y., Doherty, J.F., Van Dyck, R.E.: A Wavelet-based Watermarking Algorithm for Ownership Verification of Digital Images. IEEE Transaction on Image Processing 11(2), 77–88 (2002)
3. Kirovski, D., Malvar, H.S.: Spread Spectrum Watermarking of Audio Signals. IEEE Transactions on Signal Processing 51(4), 1020–1033 (2003)
4. Langelaar, G.C., Lagendijk, R.L.: Optimal Differential Energy Watermarking of DCT Encoded Images and Video. IEEE Transactions on Image Processing 10(1), 148–158 (2001)
5. Doncel, V.R., Nikolaidis, N., Pitas, I.: An Optimal Detector Structure for the Fourier Descriptors Domain Watermarking of 2D Vector Graphics. IEEE Transactions on Visualization and Computer Graphics 13(5), 851–863 (2007)
6. Yan, H., Li, J., Wen, H.: A Key Points-based Blind Watermarking Approach for Vector Geospatial Data. Computers, Environment and Urban Systems 35(6), 485–492 (2011)
7. Wang, C.J., Peng, Z.Y., Peng, Y.W., Yu, L., Wang, J.Z., Zhao, Q.Z.: Watermarking Geographical Data on Spatial Topological Relations. Multimedia Tools Applications 57(1), 67–89 (2012)
8. Jin, X., Zhang, Z., Wang, J., Li, D.: Watermarking Spatial Trajectory Database. In: Zhou, L., Ooi, B., Meng, X. (eds.) DASFAA 2005. LNCS, vol. 3453, pp. 56–67. Springer, Heidelberg (2005)
9. Lucchese, C., Vlachos, M., Rajan, D., Yu, P.S.: Rights Protection of Trajectory Datasets with Nearest-neighbor Preservation. The VLDB Journal 19, 531–556 (2010)
10. Sion, R., Atallah, M., Prabhakar, S.: Resilient Rights Protection for Sensor Streams. In: IEEE International Conference on Very Large Databases, vol. 30, pp. 732–743 (2004)
11. Sion, R., Prabhakar, S.: Rights Protection for Discrete Numeric Streams. IEEE Transactions on Knowledge and Data Engineering 18(5), 699–714 (2006)

12. Yuan, N.J., Zheng, Y., Zhang, L., Xie, X.: T-Finder: A Recommender System for Finding Passengers and Vacant Taxis. IEEE Transactions on Knowlege and Data Enginerring 25(10), 2390–2403 (2013)
13. Ma, S., Zheng, Y., Wolfson, O.: T-Share: A Large-Scale Dynamic Taxi Ridesharing. In: IEEE International Conference on Data Engineering, pp. 410–421 (2013)
14. Aung, H.H., Guo, L., Tan, K.-L.: Mining sub-trajectory cliques to find frequent routes. In: Nascimento, M.A., Sellis, T., Cheng, R., Sander, J., Zheng, Y., Kriegel, H.-P., Renz, M., Sengstock, C. (eds.) SSTD 2013. LNCS, vol. 8098, pp. 92–109. Springer, Heidelberg (2013)
15. Lee, J.G., Han, J., Whang, K.Y.: Trajectory Clustering: A Partition-and-Group Framework. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 593–604 (2007)
16. Muckell, J., Hwang, J.H., Lawson, C.T., Ravi, S.S.: Algorithms for Compressing GPS Trajectory Data: An Empirical Evaluation. In: Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 402–405 (2010)
17. Chen, M., Xu, M., Franti, P.: A Fast O(N) Multiresolution Polygonal Approximation Algorithm for GPS Trajectory Simplification. IEEE Transactions on Image Processing 21, 2770–2785 (2012)
18. Potamias, M., Patroumpas, K., Sellis, T.: Sampling Trajectory Streams with Spatiotemporal Criteria. In: International Conference on Scientific and Statistical Database Management, pp. 275–284 (2006)
19. Muckell, J., Hwang, J.H., Patil, V., Lawson, C.T., Ravi, S.S.: SQUISH: An Online Approach for GPS Trajectory Compression. In: International Conference on Computing for Geospatial Research & Applications, p. 13 (2011)
20. Muckell, J., Olsen, P.W., Hwang, J.H., Lawson, C.T., Ravi, S.S.: Compression of Trajectory Data: A Comprehensive Evaluation and New Approach. GeoInformatica (2013), doi:10.1007/s10707-013-0184-0
21. Keogh, E., Chu, S., Hart, D., Pazzani, M.: An Online Algorithm for Segmenting Time Series. In: Proceedings of IEEE International Conference on Data Mining, pp. 289–296 (2001)
22. Douglas, D.H., Peucker, T.K.: Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature. Canadian Cartographer 10, 112–122 (1973)
23. Lee, W.C., Krumm, J.: Trajectory preprocessing. In: Computing with Spatial Trajectories, pp. 3–33. Springer, New York (2011)
24. Schneier, B.: Applied Cryptography. John Wiley and Sons (1996)
25. Chen, B.: Quantization Index Modulation: A Class of Provably Good Methods for Digital Watermarking and Information Embedding. IEEE Transactions on Information Theory 47(4), 1423–1443 (2001)
26. Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., Huang, Y.: T-drive: Driving directions based on taxi trajectories. In: Proc. of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 99–108 (2010)
27. Yuan, J., Zheng, Y., Xie, X., Sun, G.: Driving with knowledge from the physical world. In: Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 316–324 (2011)
28. Cox, I., Miller, M., Bloom, J.: Digital watermarking. Morgan Kaufmann (2001)
29. Srinivasan, S., Dhakar, N.S.: Route-choice Modeling using GPS-based Travel Surveys (2011-008) (2013)