Spatiotemporal Compression Techniques for Moving Point Objects

Nirvana Meratnia[1] and Rolf A. de By[2]

[1] Department of Computer Science
University of Twente
P.O. Box 217, 7500 AE, Enschede
The Netherlands
meratnia@cs.utwente.nl
[2] Intl. Inst. for Geo-information Science & Earth Observation (ITC)
P.O. Box 6, 7500 AA, Enschede
The Netherlands
deby@itc.nl

Abstract. Moving object data handling has received a fair share of attention over recent years in the spatial database community. This is understandable as positioning technology is rapidly making its way into the consumer market, not only through the already ubiquitous cell phone but soon also through small, on-board positioning devices in many means of transport and in other types of portable equipment. It is thus to be expected that all these devices will start to generate an unprecedented data stream of time-stamped positions. Sooner or later, such enormous volumes of data will lead to storage, transmission, computation, and display challenges. Hence, the need for compression techniques.
Although previously some work has been done in compression for time series data, this work mainly deals with one-dimensional time series. On the other hand, they are good for short time series and in absence of noise, two characteristics not met by moving objects.
We target applications in which present and past positions of objects are important, so focus on the compression of moving object trajectories. The paper applies some older techniques of line generalization, and compares their performance against algorithms that we specifically designed for compressing moving object trajectories.

1 Database Support for Moving Objects Is Wanting

This is a crowded world with mobile inhabitants. Their mobility gives rise to traffic, which, due to various behavioural characteristics of its agents, is a phenomenon that displays patterns. It is our aim to provide tools to study, analyse and understand these patterns. We target traffic in the widest sense: commuters in urban areas (obviously), a truck fleet at the continental scale, pedestrians in shopping malls, airports or railway stations, shopping carts in a supermarket, pieces of luggage in airport logistics, even migratory animals, under the assumption that one day we will have the techniques to routinely equip many of them

E. Bertino et al. (Eds.): EDBT 2004, LNCS 2992, pp. 765–782, 2004.

with positioning devices. Fundamental in our approach is that we are not only interested in present position, but also in positional history.

In recent years, positioning technology, location-based services and ubiquitous applications have become a focus of attention in different disciplines. Perhaps, most importantly, positioning technology is becoming increasingly more available — cheaper, smaller, less power consumption — and more accurate. This development does not depend on GPS technology alone: in-house tracking technology applies various techniques for up-to-date positional awareness, and adaptable antenna arrays can do accurate positioning on cell phones [1].

Databases have not very well accommodated such data in the past, as their design paradigm was always one of 'snapshot representation'. Their present support for *spatial* time series is at best rudimentary. Consequently, database support for *moving object* representation and computing has become an active research domain. See, for instance [2,3,4,5,6].

As indicated above, there is a multitude of moving object applications. We use the phrase as a container term for various organic and inorganic entities that demonstrate mobility that itself is of interest to the application. Our principal example is urban traffic, specifically commuter traffic, and rush hour analysis. The mobile object concept, however, doesn't stop there.

Monitoring and analysing moving objects necessitates the availability of complete geographical traces to determine locations that objects have had, have or will have. Due to the intrinsic limitations of data acquisition and storage devices such inherently continuous phenomena are acquired and stored (thus, represented) in a discrete way. Right from the start, we are dealing with approximations of object trajectories. Intuitively, the more data about the whereabouts of a moving object is available, the more accurate its true trajectory can be determined. Availability of data is by no means a problem as the coming years will witness an explosion of such positional data for all sorts of moving objects. Moreover, there seem to be few technological barriers to high position sampling rates. However, such enormous volumes of data lead to storage, transmission, computation, and display challenges. Hence, there is a definite need for *compression techniques* of moving object trajectories.

Perhaps an example could help to better illustrate the essence of an effective compression technique. Let us assume that a moving object data stream is a sequence of $\langle t, x, y \rangle$, in which x, y represent coordinates of the moving object at time t, respectively. If such data is collected every 10 seconds, a simple calculation shows that 100 Mb of storage capacity is required to store the data for just over 400 objects for a single day, barring any data compression. We obviously want to be monitoring many more moving objects, and for much longer time periods.

In this paper, various compression techniques for streams of time-stamped positions are described and compared. We compare them against two new spatiotemporal compression techniques, which are described in Sect. 3.3. The superiority of the proposed methods for our application is demonstrated in Sect. 4.3. We have previously reported on this work in [7].

2 Spatial Compression Techniques

Object movement is continuous in nature. Acquisition, storage and processing technology, however, force us to use discrete representations. In its simplest form, an object trajectory is a positional time series, i.e., a (finite) sequence of time-stamped positions. To determine the object's position at time instants not explicitly available in the time series, approximation techniques are used. Piecewise linear approximation is one of the most widely used methods, due to its algorithmic simplicity and low computational complexity.

Intuitively, a piecewise linear approximation of a positional time series assumes that successive data points are connected by straight segments. There exist non-linear approximation techniques, e.g., using Bézier curves or splines [8], but we do not consider these here. In many of the applications we have in mind, object movement appears to be restricted to an underlying transportation infrastructure that itself has linear characteristics. This work, however, makes no assumption as to such positional restrictions.

Our objectives for data compression are:

- to obtain a lasting reduction in data size;
- to obtain a data series that still allows various computations at acceptable (low) complexity;
- to obtain a data series with known, small margins of error, which are preferably parametrically adjustable.

As a consequence our interest is with lossy compression techniques. The reasons for the last objective are that (i) we know our raw data to already contain error, (ii) depending on the application, we could permit additional error, as long as we understand the behaviour of error under settings of the algorithmic parameters.

Compression algorithms can be classified on another basis as well. They are either *batch* or *online* algorithms, based on whether they require the availability of the full data series. Batch algorithms do; online algorithms do not, and are typically used to compress data streams in real-time. Batch algorithms consistently produce higher quality results [9] when compared to online algorithms.

Appearing under different names, most compression algorithms relevant here can be grouped into one of the following four categories [10]:

Top-Down: The data series is recursively partitioned until some halting condition is met.

Bottom-up: Starting from the finest possible representation, successive data points are merged until some halting condition is met. The algorithm may not visit all data points in sequence.

Sliding Window: Starting from one end of the data series, a window of fixed size is moved over the data points, and compression takes place only on the data points inside the window.

Opening Window: Starting from one end of the data series, a data segment, i.e., a subseries of the data series, is grown until some halting condition is

met. Then, a compression takes place on the data points inside the window. This will decrease the window size, after which the process is continued. The 'window' size is the number of data points under consideration at any one point during the execution of the algorithm.

We have coined the term 'opening window' to do justice to the dynamic nature of the size of the window in such algorithms. They are, however, elsewhere sometimes considered 'sliding window' algorithms.

Various halting conditions may be applied. Possible conditions are:

- The number of data points, thus the number of segments, exceeds a user-defined value.
- The maximum error for a segment exceeds a user-defined threshold.
- The sum of the errors of all segments exceeds a user-defined threshold.

An obvious class of candidate algorithms for our problem are the line generalization algorithms. Some of these algorithms are very simple in nature and do not take into account any relationship between neighbouring data points. They may eliminate all data points except some specific ones, e.g., leaving in every i^{th} data point [11]. Another sort of compression algorithm utilizes the characteristics of the neighbouring data points in deciding whether to eliminate one of them. Particulary, these algorithms may use the Euclidean distance between two neighbour points. If it is less than a predefined threshold, one is eliminated. All these algorithms are sequential in nature, that is they gradually process a line from the beginning to the end.

Although these two groups of algorithms are computationally efficient, they are not so popular or widely used. Their primary disadvantage is the frequent elimination or misrepresentation of important points such as sharp angles. A secondary limitation is that straight lines are still over-represented [12], unless small differences in angle are used as another discarding condition. An algorithm to overcome the first limitation was reported by Jenks [13], and involved evaluating the perpendicular distance from a line connecting two consecutive data points to an intermediate data point against a user threshold. To tackle the second disadvantage, [14] utilized the angular change between each three consecutive data points.

The most important compression algorithms that seem to hold conceptual promise are reviewed in Sects. 2.1 and 2.2.

2.1 Top-Down Compression Algorithms

The top-down algorithms work by considering every possible split of the data series — i.e., where the approximation error is above some user-defined threshold — and selecting the best position amongst them. The algorithm then recursively continues to split the resulting subseries until they all have approximation errors below the threshold [10].

An often used and quite famous top-down method is the *Douglas-Peucker* (DP) algorithm [12]. It was originally proposed for line simplification, and tries to

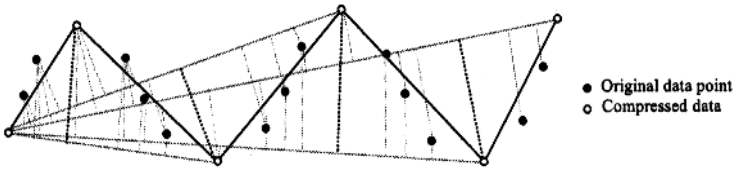

Fig. 1. Top-down Douglas-Peucker algorithm. Original data series of 19 points. In this case, only the first segment was recursively cut at data points 16, 12, 8 and 4.

preserve directional trends in the approximation line using a distance threshold, which may be varied according to the amount of simplification required. McMaster [15] who gives a detailed study of mathematical similarity and discrepancy measures, ranks the DP algorithm as 'mathematically superior'. White [16] performed a study on simplification algorithms on critical points as a psychological feature of curve similarity and showed that the DP algorithm was best at choosing splitting points; he refers to the obtained results as 'overwhelming'.

The algorithm works on the following basis. The first point of the data series is selected as the *anchor point*; the last data point is selected as the *float point*. For all intermediate data points, the (perpendicular) distance to the line connecting anchor and float points is determined. If the maximum of these distances is greater than a pre-defined threshold, the line is cut at the data point that causes that maximum distance. This cut point becomes the new float point for the first segment, and the anchor point for the second segment. The procedure is recursively repeated for both segments. The algorithm is illustrated in Fig. 1.

The DP algorithm clearly is a batch algorithm, as the whole data series is needed at the start; the time complexity of the original algorithm is $O(N^2)$ with N being the number of data points. Due to its simplicity, different implementations have been proposed. One such proposal, which succeeded to reduce the complexity of the method to $(N \log N)$ was Hershberger's proposal [17], who defined the path hull as a basis for their implementation.

2.2 Opening Window Compression Algorithms

Opening window (OW) algorithms anchor the start point of a potential segment, and then attempt to approximate the subsequent data series with increasingly longer segments. It starts by defining a segment between a first data point (the anchor) and the third data point (the float) in the series. As long as all distances of intermediate data points are below the distance threshold, an attempt is made to move the float one point up in the data series. When the threshold is going to be exceeded, two strategies can be applied: either,

- the data point causing the threshold violation (Normal Opening Window, a.k.a. NOPW), or
- the data point just before it (Before Opening Window, a.k.a. BOPW)

becomes the end point of the current segment, and it also becomes the anchor of the next segment. If no threshold excess takes place, the float is moved one up

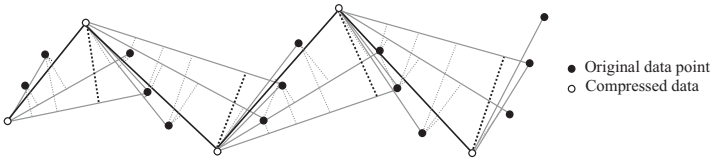

Fig. 2. Data series compression result of NOPW strategy: the threshold excess data point is the break point. The data series was broken at data points 4, 8, 12 and 16.

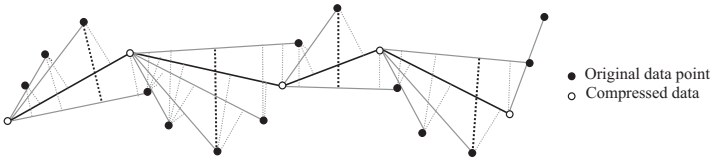

Fig. 3. Data series compression result of BOPW strategy: the data point just before the threshold excess data point is the break point. The first window opened up to point 6 (point 4 causing excess), making point 5 the cut point; second window opened up to point 11 (8 causing excess) with 10 becoming the cut point etc.

the data series — the window opens further — and the method continues, until the entire series has been transformed into a piecewise linear approximation. The results of choosing either strategy are illustrated in Figs. 2 and 3.

An important observation, which can clearly be made from Figs. 2 and 3 is that OW algorithms may lose the last few data points. Countermeasures are required.

Although OW algorithms are computationally expensive, they are popular. This is because they are online algorithms, and because they can work reasonably well in presence of noise but only for relatively short data series. The time complexity of these algorithms is $O(N^2)$.

3 Spatiotemporal Algorithms

3.1 Why Line Generalizations Do Not Quite Apply

We discussed the above algorithms because they are well-known techniques for generalizing line structures. All of them use *perpendicular distance* of data points to a proposed generalized line as the condition to discard or retain that data point. This is the mechanism at work also when we apply these algorithms to our data series, the moving object trajectories, viewed as lines in two-dimensional space.

But our trajectories have this important extra dimension, time. Intrinsically, they are not lines, but historically traced points. As a consequence, the use of perpendicular distance as condition is at least challenged, and we should look at more appropriate conditions.

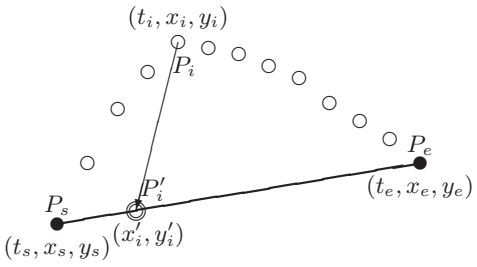

Fig. 4. Original data points (open circles), including P_i, the start and end points P_s and P_e of the approximated trajectory, and P_i's approximated position P'_i.

A trajectory is represented as a time series of positions. Generalizing a trajectory means to replace one time series of positions with another one. Like before, we can measure how effective this generalization is by looking at (a) the compression rate obtained, and (b) the error committed. Unlike before, the error committed is no longer measured through (perpendicular) distances between original data points and the new line, but rather through distances between pairs of temporally synchronized positions, one on the original and one on the new trajectory. This is a fundamental change that does justice to the spatio*temporal* characteristic of a moving point trajectory.

3.2 A Simple Class of Spatiotemporal Algorithms

The computational consequence of the above arguments is that the decision of discarding a data point must be based on its position *and* timestamp, as well as on the approximated position of the object on the new trajectory. This gives a distance not necessarily perpendicular to the new, approximated, trajectory.

The situation is illustrated in Fig. 4, in which the original data point P_i and its approximation P'_i on the new trajectory $P_s - P_e$ are indicated. The coordinates of P'_i are calculated from the simple *ratio* of two time intervals Δe and Δi, indicating respectively travel time from P_s to P_e (along either trajectory) and from P_s to P_i (along the original trajectory), respectively. These travel times are determined from the original data, as timestamp differences. We have

$$\Delta e = t_e - t_s$$
$$\Delta i = t_i - t_s$$
$$x'_i = x_s + \frac{\Delta i}{\Delta e}(x_e - x_s) \quad (1)$$
$$y'_i = y_s + \frac{\Delta i}{\Delta e}(y_e - y_s) \ . \quad (2)$$

After the approximate position P'_i is determined, the next step is to calculate the distance between it and the original P_i, and use that distance as a discarding

criterion against a user-defined threshold. This is an important improvement not only because we are using a more accurate distance measure but also because the temporal factor is now included. The continuous nature of moving objects necessitates the inclusion of *temporal* as well as *spatial* properties of moving objects.

The application of the above distance notion for moving object trajectories, in either top-down and opening window algorithms, leads to a class of algorithms that we call here *time ratio algorithms*. We will later see that under an improved error notion, this class gives substantial improvements of performance in compression rate/error trade-offs.

In the sequel, by

- *TD-TR* we denote a top-down time-ratio algorithm, obtained from the DP algorithm through application of the above time-ratio distance measuring technique, and by
- *OPW-TR* we mean a opening-window algorithm applying the same time-ratio distance measurement.

3.3 A More Advanced Class of Spatiotemporal Algorithms

Further improvements can be obtained by exploiting other spatiotemporal information hiding in the time series. Our time-ratio distance measurement was a first step; a second step can be made by analysing the derived speeds at subsequent segments of the trajectory, when these are available. A large difference between the travel speeds of two subsequent segments is another criterion that can be applied to retain the data point in the middle. For this, we will assume a *speed difference threshold* will also have been set, indicating above which speed difference we will always retain the data point.

By integrating the concepts of *speed difference threshold* and the *time-ratio distance* discussed in Sect. 3.2, we obtain a new algorithmic approach, that we call the class of *spatiotemporal algorithms*.

Observe that in principle both these concepts allow application in top-down and opening-window algorithms. We have restricted ourselves here to an opening-window version, applying both criteria. The pseudocode for the algorithm is provided below. The notation used is described in Table 1.

The algorithm sets the anchor point, and then gradually 'opens the window'. In each step, two halting conditions are verified, one on the synchronous distance measure (using the time interval ratio), the other being a difference in speed values between previous and next trajectory segment. These speeds are *not* measured speeds, as we do not assume these to be available; rather, they are speed values derived from timestamps and positions.

procedure $SPT(s, max\_dist\_error, max\_speed\_error)$
if $len(s) \leq 2$
then return s
else $is\_error \leftarrow false$
$e \leftarrow 2$

Table 1. Overview of data types, variables and functions used in algorithm SPT

$\mathbb{R}, \mathbb{N}$	the real, natural numbers
$\mathbb{T}, \mathbb{L}$	time stamps ($\mathbb{T} \cong \mathbb{R}$), locations ($\mathbb{L} \cong \mathbb{R} \times \mathbb{R}$)
$\mathbb{P}$	trajectories (paths), $\mathbb{P} \cong$ seq ($\mathbb{T} \times \mathbb{L}$)
$(x, y) : \mathbb{L}$	(easting, northing) coordinates
$p : \mathbb{P}$	a trajectory (path) p
$\dfrac{p : \mathbb{P}}{len(p) : \mathbb{N}}$	the number of data points in trajectory p.
$\dfrac{p : \mathbb{P}; 1 \leq i \leq len(p)}{p[i] : \mathbb{T} \times \mathbb{L}}$	the i^{th} data point of p, viewed as a time-stamped location
$\dfrac{p : \mathbb{P}; 1 \leq k \leq m \leq len(p)}{p[k, m] : \mathbb{P}}$	the subseries of p, starting at original index k up to and including index m
$\dfrac{d : \mathbb{T} \times \mathbb{L}}{d_{\mathrm{t}} : \mathbb{T}, d_{\mathrm{loc}} : \mathbb{L}}$	time stamp, location of the data point d
$\dfrac{q, r : \mathbb{L}}{dist(q, r) : \mathbb{R}}$	A function that takes two locations q, r and returns the distance between them
$\dfrac{p, s : \mathbb{P}}{p +\!\!+ s : \mathbb{P}}$	A function that concatenates two trajectories

while $e \leq len(s)$ **and not** *is_error* **do**
  $i \leftarrow 2$
  while ($i < e$ **and not** *is_error*) **do**
    $\Delta e \leftarrow s[e]_{\mathrm{t}} - s[1]_{\mathrm{t}}$
    $\Delta i \leftarrow s[i]_{\mathrm{t}} - s[1]_{\mathrm{t}}$
    $(x'_i, y'_i) \leftarrow s[1]_{\mathrm{loc}} + (s[e]_{\mathrm{loc}} - s[1]_{\mathrm{loc}})\, \Delta i / \Delta e$
    $v_{i-1} \leftarrow dist(s[i]_{\mathrm{loc}}, s[i-1]_{\mathrm{loc}}) \,/\, (s[i]_{\mathrm{t}} - s[i-1]_{\mathrm{t}})$
    $v_i \leftarrow dist(s[i+1]_{\mathrm{loc}}, s[i]_{\mathrm{loc}}) \,/\, (s[i+1]_{\mathrm{t}} - s[i]_{\mathrm{t}})$
    if $dist(s[i]_{\mathrm{loc}}, (x'_i, y'_i)) > max\_dist\_error$ **or** $\|v_i - v_{i-1}\| > max\_speed\_error$
      then $is\_error \leftarrow true$
      else $i \leftarrow i + 1$
    end if
  end while
  if *is_error*
    then return $\big[s[1]\big] +\!\!+ SPT(s[i, len(s)], max\_dist\_error, max\_speed\_error)$
  end if
  $e \leftarrow e + 1$
end while
if not *is_error*
  then return $\big[s[1], s[len(s)]\big]$
end if
end if

Table 2. Statistics on the ten moving object trajectories used in our experiments

statistic	*average*	*standard deviation*
duration	00:32:16	00:14:33
speed	40.85 km/h	12.63 km/h
length	19.95 km	12.84 km
displacement	10.58 km	8.97 km
# of data points	200	100.9

4 Comparisons and Results

To assess their appropriateness and relative performance, both spatial and spatiotemporal compression techniques were tested using real moving object trajectory data. In total, we obtained 10 trajectories through a GPS mounted on a car, which travelled different roads in urban and rural areas. The data includes short and lengthy time series; various statistics of our data set are provided in Table 2.

In using lossy compression techniques there is always a trade-off between compression rate achieved and error allowed. In our case, the optimal compression technique should find a subseries of the original time series that has a high enough compression and a low enough error. This error notion is the main tool to evaluate the quality of the compression technique. But the literature has not paid a lot of attention to explicitly measuring error. Mostly, attention has been given to the identification of proper heuristics for discarding data points.

Such heuristics may or may not be proper for compressing moving object trajectories. One of the contributions of this paper is the introduction of an error formula for moving object trajectory compression; it is described below. Since writing up this paper, we have found a similar approach was published by Nanni [18], though in a more general setting, and thus not providing the full details of the formula that we are after.

4.1 Error Notions

The quality of compression techniques is evaluated by using some notion of error. Several mathematical measures have been proposed for this error notion, for instance by [15,19]. Error notions can be based on different principles: length, density, angularity and curvilinearity. Some of these notions have been used to improve the appeal of the approximation (usually a line) in cartographic context.

Distance-based error notions seem less biased towards visual effect. In plain line generalization, perpendicular distance is the error measure of choice. A simple method determines all distances of original data points to the approximation line, and determines their average, but this is sensitive to the actual number of data points. A method least sensitive to this number essentially determines the area between original line and approximation. It effectively assumes there are infinitely many original data points.

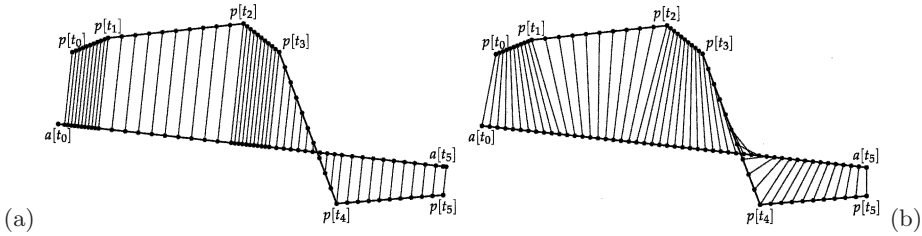

Fig. 5. Two error notions for a trajectory p being approximated by a. (a) error measured at fixed sampling rate as sum of perpendicular distance chords; (b) error measured at fixed sampling rates as sum of time-synchronous distance chords.

How would such an error measure translate to the spatiotemporal case of object movement? Applying still perpendicular distances, insensitivity to the number of data points can be obtained by considering progressively finer sampling rates, as illustrated in Fig. 5a. In this figure, p represents an original trajectory with five segments, a represents its 1-segment approximation. The t_i are equally spaced time instants. On slower segments (like the first and third), distance chords become more condensed. For progressively finer sampling rates, this error notion becomes the sum over segments of weighted areas between original and approximation. The associated formulas are simple and direct.

4.2 A Spatiotemporal Error Notion

Given an original trajectory $(p : \mathbb{P})$ and an approximation trajectory $(a : \mathbb{P})$ of it, we are interested in finding a measure that expresses without bias how well the second approximates the first. The plain intuition we have for this is that the *average distance between the original object and the approximation object*—both *synchronously* travelling along their respective trajectories p and a—during the time interval of their trajectories is the best measure that one can have. In the remainder of this section, we are developing the formulas for that average distance.

We will assume that both trajectories (p and a) are represented as series of time-stamped positions, which we will interpret as piecewise linear paths. Given the classes of compression algorithms that we study, we can also safely assume that the time stamps present in the trajectory a form a subseries of those present in the original p. After all, we have obtained that series by discarding data points in the original, and we never invented new data points, let alone time stamps. Assume that p has data points numbered from 1 to k.

Following from the above, we can define the *average synchronous error* $\alpha(p, a)$ between p and a as a summation of the weighted contributions of all linear segments in p.

$$\alpha(p,a) = \frac{\sum_{i=1}^{k-1} (p[i+1]_{\mathrm{t}} - p[i]_{\mathrm{t}}) \cdot \alpha(p[i:i+1], a)}{\sum_{i=1}^{k-1} p[i+1]_{\mathrm{t}} - p[i]_{\mathrm{t}}} \quad . \tag{3}$$

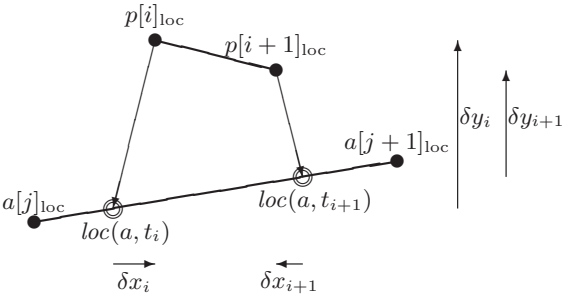

Fig. 6. Definition of $\delta x_i, \delta x_{i+1}, \delta y_i$ and δy_{i+1}. Illustrated is the i-th segment of original path p (at top), between points $p[i]_{\text{loc}}$ and $p[i+1]_{\text{loc}}$, and its approximation $loc(a, t_i)$–$loc(a, t_{i+1})$ on trajectory a. Observe that this approximation will be part of an a-segment, but that its start and end points may or may not coincide with real data points of a.

We have not defined the function α fully in this way. Equation 3 covers the case that p is a multi-segment trajectory, but not the case that it is a single segment. We will cover that case through (4) below.

Now let us derive the *single segment average synchronous error*. If q is a single segment trajectory, we can define $loc(q, t)$ as the moving object position at time t, in the style of (1) and (2). We generalize this notion of object position for an arbitrary length trajectory p in the obvious way, such that $loc : \mathbb{P} \to (\mathbb{T} \to \mathbb{L})$ is a total function such that for any p, $loc(p)$ is a partial function with domain $[p[1]_{\text{t}}, p[len(p)]_{\text{t}}]$.

With the single segment average synchronous error $\alpha(p[i : i+1], a)$ we express the average distance between original and approximate object during the time interval between indices i and $i+1$, for convenience written as $[t_i, t_{i+1}]$. It can be expressed as

$$\alpha(p[i : i+1], a) = \frac{1}{t_{i+1} - t_i} \int_{t_i}^{t_{i+1}} dist(loc(p, t), loc(a, t))\, dt \ . \tag{4}$$

In our further derivations, differences in coordinate values at time instants t_i and t_{i+1} between p and a (in that order) will be important. We will denote these (four) differences, respectively as $\delta x_i, \delta x_{i+1}, \delta y_i$ and δy_{i+1}. E.g., δy_i equals $loc(p, t_i).y - loc(a, t_i).y$, and so on. Observe that these are constants; they are illustrated in Fig. 6.

Since both p and a during the given time interval are represented as straight segments, the distance formula in (4) can be derived as having a well-known polynomial format:

$$dist(loc(p, t), loc(p, t)) = \frac{1}{c_4} \sqrt{c_1 t^2 + c_2 t + c_3},$$

where

$$c_1 = (\delta x_i - \delta x_{i+1})^2 + (\delta y_i - \delta y_{i+1})^2$$

$c_2 = 2 \cdot ((\delta x_{i+1} t_i - \delta x_i t_{i+1}) \cdot (\delta x_i - \delta x_{i+1}) + (\delta y_{i+1} t_i - \delta y_i t_{i+1}) \cdot (\delta y_i - \delta y_{i+1}))$
$c_3 = (\delta x_{i+1} t_i - \delta x_i t_{i+1})^2 + (\delta y_{i+1} t_i - \delta y_i t_{i+1})^2$ and
$c_4 = t_{i+1} - t_i$.

Using the above results we can rewrite (4) to

$$\alpha(p[i:i+1], a) = \frac{1}{(t_{i+1} - t_i)^2} \int_{t_i}^{t_{i+1}} \sqrt{c_1 t^2 + c_2 t + c_3}\, dt \ . \tag{5}$$

The solution to (5) depends on the values for the constants c_i. We provide a case analysis, omitting cases that cannot happen due to the structure—expressed in the equations for constants c_i—of the problem.

Case $c_1 = 0$ **:** This happens when $\delta x_i = \delta x_{i+1} \wedge \delta y_i = \delta y_{i+1}$. If so, then also $c_2 = 0$, and the solution to (5) is

$$\alpha(p[i:i+1], a) = \frac{\sqrt{c_3}}{t_{i+1} - t_i} \ .$$

The geometric interpretation of this case is simple: equality of δ's indicates that the approximation of this p segment is a vector-translated version of that segment. The distance is thus constant, and its average over the time interval equals that constant. We have:

$$\boxed{\alpha(p[i:i+1], a) = \sqrt{\delta x_i^2 + \delta y_i^2} \ .}$$

Case $(c_1 > 0)$ **:** This happens when $\delta x_i \neq \delta x_{i+1} \vee \delta y_i \neq \delta y_{i+1}$. The solution to the integral part of (5) is non-trivial, and deserves a case analysis in itself. We have the following cases:

Case $c_2^2 - 4c_1c_3 = 0$ **:** In other words, the determinant of the quadratic subformula of (5) equals 0. This happens only when $\delta x_i \delta y_{i+1} = \delta x_{i+1} \delta y_i$. If so, the solution to (5) is

$$\boxed{\alpha(p[i:i+1], a) = \frac{1}{(t_{i+1} - t_i)^2} \left| \frac{2c_1 t + c_2}{4c_1} \sqrt{c_1 t^2 + c_2 t + c_3} \right|_{t_i}^{t_{i+1}} \ .}$$

The geometric interpretation of this case follows from the δ product equality mentioned above. That equality holds in three different cases. These cases are not mutually exclusive, but where they are not, the formulas coincide value-wise.

Case segments share start point : Consequently, we have $\delta x_i = \delta y_i = 0$. The above formula simplifies to

$$\boxed{\alpha(p[i:i+1], a) = \tfrac{1}{2} \sqrt{\delta x_{i+1}^2 + \delta y_{i+1}^2} \ .}$$

Case segments share end point : Consequently, we have $\delta x_{i+1} = \delta y_{i+1} = 0$. The above formula simplifies to

$$\boxed{\alpha(p[i:i+1],a) = \tfrac{1}{2}\sqrt{\delta x_i^2 + \delta y_i^2}\ .}$$

Case δ ratios respected : It turns out that under this condition, the synchronous distance chords (indicated as grey edges in Figs. 5) all lie parallel to each other, and this simplifies the related formulas.

Case $c_2^2 - 4c_1c_3 < 0$: The determinant of the quadratic subformula of (5) is less than 0. This is the general, non-exceptional case. The resulting formula is:

$$\boxed{\begin{array}{l}\alpha(p[i:i+1],a) = \frac{1}{(t_{i+1}-t_i)^2}\,|F(t)|_{t_i}^{t_{i+1}} \qquad \text{where} \\ F(t) = \frac{2c_1t+c_2}{4c_1}\sqrt{c_1t^2+c_2t+c_3} - \frac{c_2^2-4c_1c_3}{8c_1\sqrt{c_1}}\operatorname{arcsinh}\left(\frac{2c_1t+c_2}{\sqrt{4c_1c_3-c_2^2}}\right)\ .\end{array}}$$

4.3 Experimental Results

As mentioned earlier, all the compression techniques were tested on real data as summarized in Table 2; i.e., ten different trajectories, for fifteen different spatial threshold values ranging from 30 to 100 m, and three speed difference threshold values ranging from 5 to 25 m/s. The obtained results for each experiment consist of error produced and compression rate achieved. We used the time synchronous error notion derived in the previous paragraph. It is important to note that the choice of optimal threshold values is difficult and might differ for various applications.

A first experiment concerned the comparison between conventional top-down (Normal) DP (NDP) and our top-down time ratio (TD-TR) algorithm. Fig. 7 shows the results. Clearly, the TD-TR algorithm produces much lower errors, while the compression rate is only slightly lower.

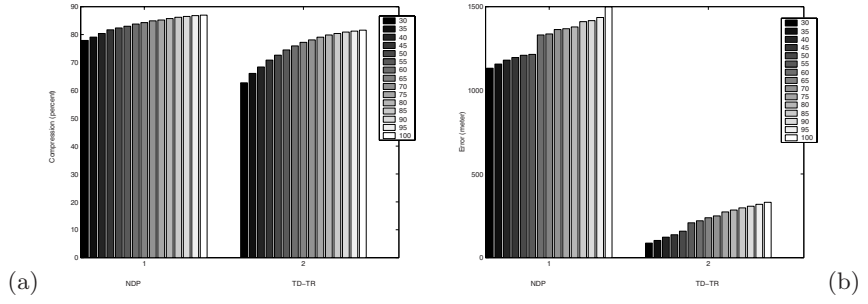

Fig. 7. Comparison between NDP (on left) and TD-TR (on right) algorithms. Colour bars indicate different settings for distance threshold, ranging from 30 to 100 m. (a) Comparison of compression percentages achieved; (b) Comparison of errors committed. Figures given are averages over ten different, real trajectories.

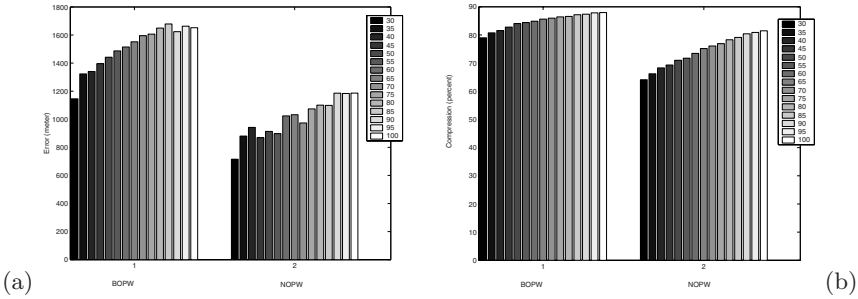

Fig. 8. Comparison between two opening window algorithms, BOPW (on left) and NOPW (on right) algorithms. For other legend information, see Fig. 7.

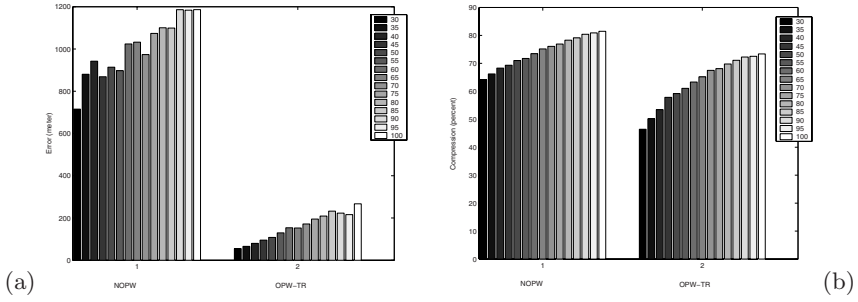

Fig. 9. Comparison between NOPW (on left) and OPW-TR (on right) algorithms. For other legend information, see Fig. 7.

An important observation is that both compression rate and error *monotonically* increase with distance threshold, asymptotically reaching a maximum.

The second experiment concerned the choice of break point in opening window algorithms; i.e., whether to break at the data point causing threshold excess (NOPW) or at the one just before (BOPW). We found that BOPW results in higher compression but worse errors. It can be used in applications where the main concern is compression and one is willing to favour it over error. For most of our application domains, we aim at lower error and high enough compression rate, and will therefore ignore the BOPW method. Results of comparison between BOPW and NOPW algorithms are shown in Fig. 8.

One may observe that error in the case of the NOPW algorithm does not strictly monotonically increase with increasing distance threshold values. We have reason to believe that these effects are systematic; they are likely only an artifact caused by the small set of trajectories in our experiments.

In Fig. 9, we illustrate the findings of a third experiment, comparing our opening window time ratio with a normal opening window algorithm. It demonstrates the superiority of the first (OPW-TR) with respect to the latter (NOPW) algorithm. As can be seen, for OPW-TR a change in threshold value does not dramatically impact error level. Therefore, unlike the NOPW algorithm, in which a choice of threshold is important for the error results, in the OPW-TR algorithm

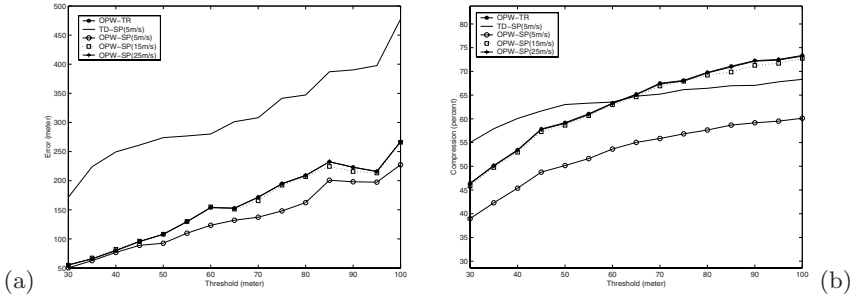

Fig. 10. Comparison between OPW-TR, TD-SP and OPW-SP algorithms. (a) Errors committed; (b) Compression obtained. In both figures, the graph for OPW-TR coincides with that of OPW-SP-25m/s.

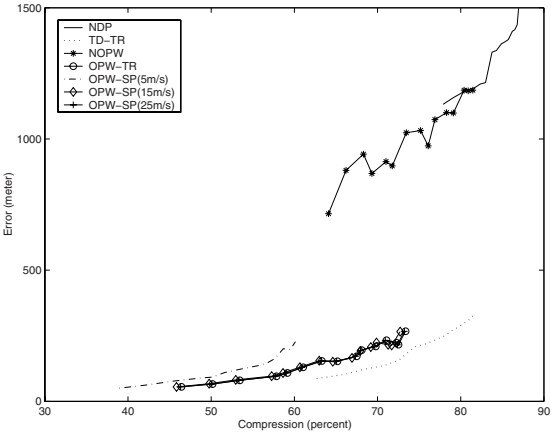

Fig. 11. Error versus Compression in NDP, TD-TR, NOPW, OPW-TR and OPW-SP algorithms. As before, the graph for OPW-TR coincides with that of OPW-SP-25m/s.

this is not the case. This allows choosing a higher threshold value to improve compression while not loosing much on error performance.

Our second spatio-temporal algorithm (SP) was applied in both opening window and top-down fashion. In case of top-down SP (TD-SP), experiments show a high sensitivity towards speed threshold settings, both for error and compression. Among the three speed difference threshold values (5, 15, 25 m/s), only the first value provided reasonable results, and is therefore included in the comparisons. In opening window fashion SP (OPW-SP) changes in threshold value did not lead to dramatic changes in the results. This is an important characteristic of both OPW-TR and OPW-SP algorithms. As can be seen from Fig. 10, the two OPW-SP (15 m/s, 25 m/s) algorithms as well as OPW-TR have very similar behaviour in both compression rate and error. On the other hand, choosing a speed difference threshold of 5 m/s in TD-SP and OPW-SP results in improved compression as well as higher error in TD-SP.

As experiments show reasonably high compression rates as well as low errors for our TD-TR, OPW-TR and OPW-SP algorithms, they effectively fulfill the

requirement of *good* compression techniques. A final comparison between these algorithms ranks the TD-TR slightly over their counterparts because of better compression rate. See Fig. 11.

However, two issues should not be forgotten. One is that TD-TR is a batch algorithm, while OPW-TR and OPW-SP are online algorithms. The other issue is that the higher compression rate in TD-TR is accompanied by slightly higher error. Therefore, depending on data availability and error allowed, any of the mentioned algorithms can be used. The results of this final comparison are shown in Fig. 11. It clearly shows that algorithms developed with spatiotemporal characteristics outperform others. Another important observation is that the choice of threshold value for OPW-SP is crucial, as it may rank it higher or lower than OPW-TR.

5 Conclusions and Future Work

Existing compression techniques commonly used for line generalization are not suitable for moving object trajectory applications. Mostly because they operate on the basis of perpendicular distances. Due to the continuous nature of moving objects, both spatial and temporal factors should be taken into account compressing their data.

In this paper, problems of existing compression techniques for moving object application are addressed. Two spatio-temporal techniques are proposed to overcome the mentioned problems. The quality of the methods was tested using a new and advanced error notion. The experiments confirm the superiority of the proposed methods to the existing ones. The proposed algorithms are suitable to be used as both *batch* and *online*.

Obtained results strongly depend on the chosen threshold values. Choosing a proper threshold is not easy and is application-dependent. However, having a clear understanding of moving object behaviour helps in making these choices, and we plan to look into the issue of moving objects of different nature.

Piecewise linear interpolation was used as the approximation technique. Considering that other measurements such as momentaneous speed and direction values are sometimes available, other, more advanced, interpolation techniques and consequently other error notions can be defined. This is an issue that we want to address in the future.

Acknowledgements. We thank the anonymous reviewers for their elaborate and detailed comments that helped us to improve the paper.

References

1. Cooper, M.: Antennas get smart. Scientific American **283** (2003) 48–55
2. Abdelguerfi, M., Givaudan, J., Shaw, K., Ladner, R.: The 2-3TR-tree, a trajectory-oriented index structure for fully evolving valid-time spatio-temporal datasets. In: Proc. 10th ACM-GIS, ACM Press (2002) 29–34

3. Zhu, H., Su, J., Ibarra, O. H.: Trajectory queries and octagons in moving object databases. In: Proc. 11th CIKM, ACM Press (2002) 413–421
4. Güting, R. H., Böhlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N. A., Schneider, M., Vazirgiannis, M.: A foundation for representing and querying moving objects. ACM TODS **25** (2000) 1–42
5. Šaltenis, S., Jensen, C. S., Leutenegger, S. T., Lopez, M. A.: Indexing the positions of continuously moving objects. In: Proc. ACM SIGMOD, ACM Press (2000) 331–342
6. Agarwal, P. K., Guibas, L. J., Edelsbrunner, H., Erickson, J., Isard, M., Har-Peled, S., Hershberger, J., Jensen, C., Kavraki, L., Koehl, P., Lin, M., Manocha, D., Metaxas, D., Mirtich, B., Mount, D., Muthukrishnan, S., Pai, D., Sacks, E., Snoeyink, J., Suri, S., Wolfson, O.: Algorithmic issues in modeling motion. ACM Computing Surveys **34** (2002) 550–572
7. Meratnia, N., de By, R. A.: A new perspective on trajectory compression techniques. In: Proc. ISPRS DMGIS 2003, October 2–3, 2003, Québec, Canada. (2003) S.p.
8. Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F.: Computer Graphics: Principles and Practice. Second edn. Addison-Wesley (1990)
9. Shatkay, H., Zdonik, S. B.: Approximate queries and representations for large data sequences. In Su, S.Y.W., ed.: Proc. 12th ICDE, New Orleans, Louisiana, USA, IEEE Computer Society (1996) 536–545
10. Keogh, E. J., Chu, S., Hart, D., Pazzani, M. J.: An online algorithm for segmenting time series. In: Proc. ICDM'01, Silicon Valley, California, USA, IEEE Computer Society (2001) 289–296
11. Tobler, W. R.: Numerical map generalization. In Nystuen, J.D., ed.: IMaGe Discussion Papers. Michigan Interuniversity Community of Mathematical Geographers. University of Michigan, Ann Arbor, Mi, USA (1966)
12. Douglas, D. H., Peucker, T. K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer **10** (1973) 112–122
13. Jenks, G. F.: Lines, computers, and human frailties. Annuals of the Association of American Geographers **71** (1981) 1–10
14. Jenks, G. F.: Linear simplification: How far can we go? Paper presented to the Tenth Annual Meeting, Canadian Cartographic Association (1985)
15. McMaster, R. B.: Statistical analysis of mathematical measures of linear simplification. The American Cartographer **13** (1986) 103–116
16. White, E. R.: Assessment of line generalization algorithms using characteristic points. The American Cartographer **12** (1985) 17–27
17. Hershberger, J., Snoeyink, J.: Speeding up the Douglas-Peucker line-simplification algorithm. In: Proc. 5th SDH. Volume 1., Charleston, South Carolina, USA, University of South Carolina (1992) 134–143
18. Nanni, M.: Distances for spatio-temporal clustering. In: Decimo Convegno Nazionale su Sistemi Evoluti per Basi di Dati (SEBD 2002), Portoferraio (Isola d'Elba), Italy. (2002) 135–142
19. Jasinski, M.: The compression of complexity measures for cartographic lines. Technical report 90–1, National Center for Geographic Information and Analysis, Department of Geography. State University of New York at Buffalo, New York, USA (1990)