A New Perspective On Trajectory Compression Techniques

Nirvana Meratnia
Department of Computer Science
University of Twente
P.O. Box 217, 7500 AE, Enschede
The Netherlands
meratnia@cs.utwente.nl

Rolf A. de By
Intl. Inst. for Geo-information Science
& Earth Observation (ITC)
P.O. Box 6, 7500 AA, Enschede
The Netherlands
deby@itc.nl

Abstract

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 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.

Previously some work has been done in data compression for GIS and mainly in line generalization area considering two dimensional data. However, time-stamped positions data do not form a line, as they are historically traced points. On the other hand, researches in compression for time series data mainly deal with one dimensional time series and are good for short time series and in absence of noise. This paper addresses the drawbacks of existing compression techniques and describes different techniques to compress short as well as lengthy stream of time-stamped positions data.

1 Introduction

Moving object representation and computing have 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 devices in many means of transport and in 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.

There exists a multitude of moving object application possibilities. 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, does not stop there.

Monitoring 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. 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.

In essence, *compression techniques* aim at substantial reductions in the amount of data without serious information loss. Lossless compressions have no information loss and are often based on optimizing the information capacity per byte used. Lossy compressions do suffer from information loss, and are often based on discarding the least informative data. The central theme of this paper concerns a lossy compression technique for moving object data streams that attempts to preserve the major characteristics of the original trajectory.

In this paper, various compression techniques for short and lengthy streams of time-stamped positions are described and compared. The background of these techniques is briefly addressed in Section 2. One of the most commonly used compression techniques is included in our discussions. We describe three new spatio-temporal compression techniques in Section 3. The superiority of the proposed methods is shown in Section 4, which is followed by Section 5 to conclude the paper and provide recommendations for future work.

2 Background

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. There exist non-linear approximation techniques, e.g., using Bézier curves or splines [2], 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.

One of our objectives in compression is to obtain a data series with known, small margins of error, which are preferably parametrically adjustable. Therefore, We are interested in lossy compression techniques. The reasons for this is 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 behavior of error under settings of the algorithmic parameters. Setting the parameters of a compression algorithm means always making a trade-off between the compression rate achieved and the errors allowed. Trade-off characteristics of different algorithms, as we shall see, may differ tremendously.

Appearing under different names and with slightly different implementation details, most compression algorithms for data series of the type that we consider here can be grouped into one of the following three categories [6]:

Naïve : All data points in the time series are eliminated except some specific ones.

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.

Sliding Window : Starting from one side, a data segment, i.e., a subseries, is grown until some halting condition is met. The process continues with the next data point not included in the newly approximated segment, until the end of the original series is reached. The 'window' width is the number of data points under consideration at any point during the execution of the algorithm.

Various halting conditions can be applied but algorithms do not always support these conditions in combination. Possible conditions are:

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

Some compression 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, where i is a fixed integer based upon the desired degree of compression[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 [1], unless small differences in angle are used as another discarding condition.

An algorithm to overcome the first limitation was first reported by Lang [7]. In this method, the perpendicular distance from a line connecting the first data point (the anchor) and the third data point

(the float) in the series to an intermediate data point is evaluated against a pre-defined user threshold. As long as all distances of intermediate data points, i.e., those between anchor and float, are below a distance threshold, an attempt is made to move the float one point up in the data series. When the threshold threatens to become exceeded, two strategies can be applied: either, the intermediate data point with a threshold violation or the data point just before it becomes the end point of the current segment, and it also becomes the anchor of the next segment. If no threshold excess would take place, the float is moved one up the data series, and the method continues, until the entire series has been transformed into a piecewise linear approximation. Jenks [4] basically applied the same idea on every two consecutive points.

To tackle the second disadvantage [5] utilizes the angular change between each three consecutive data points. In this method, the angle between the vector connecting the first and the third point is calculated. If this angle is greater than a pre-defined angular threshold, the intermediated point is retained. Otherwise, the intermediate point is ignored. This algorithm is likely to delete important points along the line with small curvature [8].

In general, existing compression techniques have three drawbacks as far as moving point object representations are concerned. First, they are good in absence of noise, but moving object data is noisy. Secondly, they have been developed for two dimensional data in general and not necessarily time series. While, moving object time series are 2D time series at the least and can often be extended to 3D and even higher dimensions. Last but not least, the error notions that came with the original algorithms (and their stated purposes) are not entirely appropriate for application on moving object trajectories.

In Section 2.1, one of the major compression algorithm that seems to hold conceptual promise is reviewed, in which its advantages and disadvantages are addressed and used to propose more efficient and suitable compression techniques for moving object trajectories.

2.1 Top-Down Compression Algorithms

The top-down algorithms work by considering every possible first step in partitioning the data series and splitting it at the best position if the approximation error is above some user-defined threshold. The algorithm then recursively continues to split the subseries until they all have approximation errors below the threshold [6].

One of the most famous top-down methods that is commonly used is the *Douglas-Poiker* (DP) algorithm [1]. It was originally proposed for line simplification, and tries to preserve directional trends in the approximation line using a pre-defined distance threshold, which may be varied according to the amount of simplification required.

McMaster [9] who gives a detailed study of mathematical similarity and discrepancy measures, ranks the DP algorithm as 'mathematically superior'. White [12] performed a study on simplification algorithms on critical points as a psychological of curve similarity and showed that DP method was best at choosing splitting point and called the obtained results as 'overwhelming'. [10] calls the DP algorithm as one of the most geometrically efficient algorithms in processing strings of x-y coordinate pairs. However, [8] describes the method as highly time consuming.

The algorithm works on the following basis. The first data point of the time series is selected as the *anchor point* and 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 corresponding to that maximum distance. This 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. It is illustrated in Figure 1.

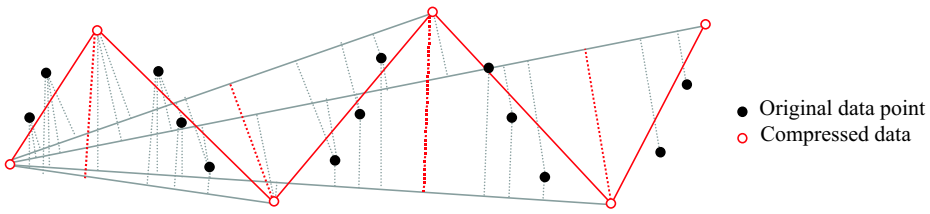

Figure 1: Douglas-Poiker algorithm

To use this algorithm, the whole data series is needed at start and the time complexity of the original algorithm is $O(N^2)$. However, due to simplicity of the algorithm, different methods have been proposed to implement it. One of such proposals, which succeed to reduce the complexity of the method from $O(N^2)$ to $(Nlog_2N)$ was Hershberger *et al*'s proposal [3], who defined the path hull as a basis for their implementation.

2.2 Missing Concept In Line Generalization Algorithms

We discussed the above algorithm because it is well-known technique for generalizing line structures, in which the *perpendicular distance* of data points to a proposed generalized line is used as the condition to discard or retain that data point. This is the mechanism at work also when we apply the algorithm 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.

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 Spatio-temporal Algorithms

3.1 A Top-Down Speed-Based Algorithm

The first improvement in the existing compression techniques can be obtained by exploiting the spatiotemporal information hiding in the time series. This can be made by analysing the derived speeds at subsequent segments of the trajectory. A large difference between the travel speeds of two subsequent segments is a criterion that can be applied to retain the data point in the middle. For this, we will assume a *speed threshold* will have been set. The data point in which speed difference is the greatest is identified. If such speed difference is greater than the pre-defined speed threshold we will always retain the data point. The application of the speed threshold notion for moving object trajectories, leads to a class of algorithms that we call here *speed-based algorithm*. In the sequel, by *TD-SP* we denote a top-down speed-based algorithm, obtained from the DP algorithm through application of the above speed threshold.

The pseudocode for the algorithm is provided below. The notation used is described in Table 1.

procedure $TD\_SP(s, speed\_threshold)$
if $len(s) \leq 2$ **then**
  return s
else $max\_speed\_threshold \leftarrow 0$
  $found\_index \leftarrow 0$
  for $i \leftarrow 2$ **until** $len(s) - 1$ **do**
    $v_{i-1} \leftarrow dist(s[i]_{\text{loc}}, s[i-1]_{\text{loc}})$ / $(s[i]_{\text{t}} - s[i-1]_{\text{t}})$
    $v_i \leftarrow dist(s[i+1]_{\text{loc}}, s[i]_{\text{loc}})$ / $(s[i+1]_{\text{t}} - s[i]_{\text{t}})$
    if $\|v_i - v_{i-1}\| > max\_speed\_threshold$ **then**
      $max\_speed\_threshold \leftarrow \|v_i - v_{i-1}\|$
      $found\_index \leftarrow i$
    end if
  end for
  if $max\_speed\_threshold > speed\_threshold$ **then**
    return $TD\_SP(s[1, found\_index], speed\_threshold) +\!\!+ TD\_SP(s[found\_index, len(s)], speed\_threshold)$
  else
    return $[s[1], s[len(s)]]$
  end if
end else

3.2 A Top-Down Time-Ratio Algorithm

The computational consequence of the arguments mentioned in 2.2 is that the decision of discarding a data point must be based on its timestamp and position, as well as on the approximated position of the object on the new trajectory. This gives a distance not necessarily perpendicular to the new, approximation trajectory.

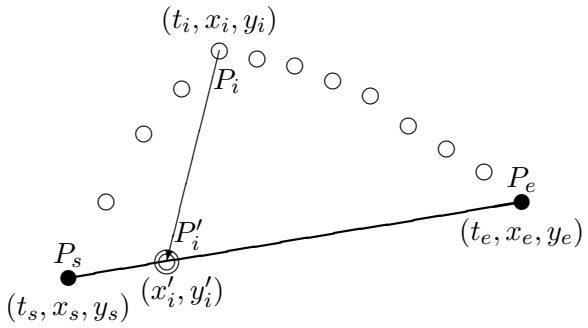

Figure 2: Original data points (∘), including P_i, start and end points P_s and P_e of the new trajectory, and P_i's approximated position P'_i.

The situation is illustrated in Figure 2, 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 simple *ratios* of two time intervals Δe and Δi, indicating travel time from P_s to P_e and from P_s to P_i, respectively. These travel times are determined from the original data, as timestamp differences.

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

After the approximate position P'_i is determined using the formulæ above, the next step is to calculate the distance between it and the original P_i, and use such a distance as a discarding criterion with 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, leads to a class of algorithms that we call here *time ratio algorithms*. We will later see 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.

The pseudocode for the algorithm is provided below. The notation used is described in Table 1.

procedure $TD\_TR(s, dist\_threshold)$
if $len(s) \leq 2$ **then**
  return s
else $max\_dist\_threshold \leftarrow 0$
  $found\_index \leftarrow 0$
  $\Delta e \leftarrow s[len(s)]_{\mathrm{t}} - s[1]_{\mathrm{t}}$
  for $i \leftarrow 2$ **until** $len(s) - 1$ **do**
    $\Delta i \leftarrow s[i]_{\mathrm{t}} - s[1]_{\mathrm{t}}$
    $(x'_i, y'_i) \leftarrow s[1]_{\mathrm{loc}} + (s[e]_{\mathrm{loc}} - s[1]_{\mathrm{loc}})\, \Delta i / \Delta e$

```
        if dist(s[i]_loc, (x'_i, y'_i)) > max_dist_threshold then
            max_dist_threshold ← dist(s[i]_loc, (x'_i, y'_i))
            found_index ← i
        end if
    end for
    if max_dist_threshold > dist_threshold then
        return TD_TR(s[1, found_index], dist_threshold) ++ TD_TR(s[found_index, len(s)], dist_threshold)
    else
        return [s[1], s[len(s)]]
    end if
end else
```

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

Table 1: Overview of data types, variables and functions used in TD-TR and TD-SP algorithms

3.3 A More Advanced Spatio-Temporal Algorithm

By integrating the concepts of *speed threshold* and the *time-ratio distance* discussed in Sections 3.1 and 3.2, we obtain a new algorithmic approach, that we call the class of *full spatiotemporal algorithms*. An important observation is that in this class of algorithms, we deal with two thresholds and therefore have to make a choice which one to exploit first. This decision means applying either (TD-SP) or (TD-TR) algorithm first, which is then augmented by the other algorithm. As we will shortly see, the results of the two might significantly differ. For the sake of clarity, we call the algorithm obtained from applying TD-TR first and then TD-SP, TD-TR-SP and the algorithm obtained from applying TD-SP first and then TD-TR, TD-SP-TR.

4 Comparisons and Results

The proposed compression techniques and conventional DP algorithm 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.

We used multiple trajectories to improve the statistical relevance of our comparisons. The original trajectory data were all time series with elements in the form of $\langle t, x, y \rangle$.

First, for each time series the original data was used to determine its corresponding trajectory. To do so, piecewise linear approximation was utilized. Depending on the trip, each trajectory has its own behaviour, ranging from simple one-directional trajectory to more complicated ones consisting of several loops, bi- or multi-directional trajectories. Moving objects like cars are subject to sudden stops, abrupt changes of direction and multiple visits to the same location in case of bi- or multi-directional trajectories. Therefore, the type of the trip may influence the performance of the compression techniques in use. Our experimentation data set had such non-trivial examples.

One trajectory and results of applying the five mentioned algorithms on it are shown in Figure 3. The behaviour is typical for many of tested moving object trajectories.

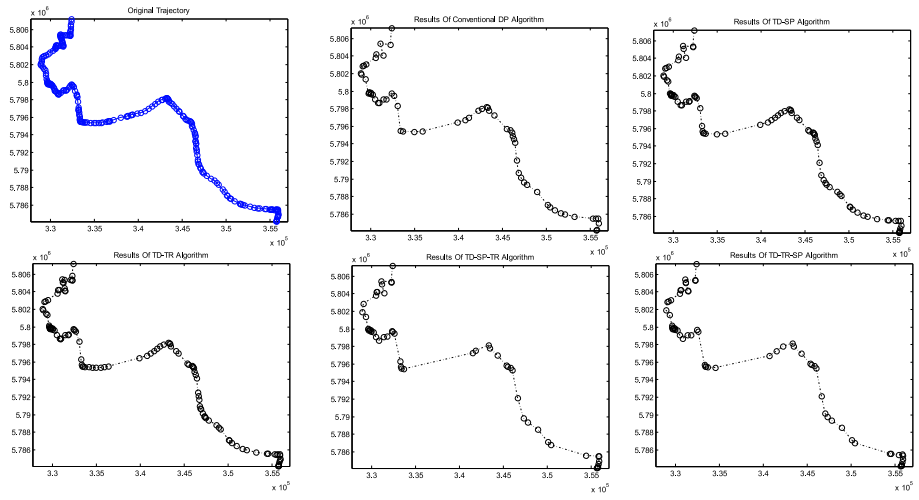

Figure 3: Original trajectory (top left) and results of applying conventional DP (top middle), TD-SP (top right), TD-TR (bottom left), TD-SP-TR (bottom middle) and TD-TR-SP (bottom right) algorithms

The superiority of our spatio-temporal methods can clearly be seen in Figure 4, in which a comparison between the five methods are made. One can observe that TD-TR can be used stand alone. However, although a noticeable improvement is made by applying speed threshold only, obtained results are significantly enhanced by integrating the method with TD-TR algorithm. It is also shown that results are enhanced by applying TD-SP before TD-TR.

Due to space limitations in this paper, unfortunately we can not show all the obtained results here and refer interested people to an upcoming, extended version of the paper.

5 Conclusions and Future Work

Existing compression techniques are not suitable for moving object trajectory applications. They are good for short time series and in absence of noise, which is definitely not the case for moving objects. Their other main drawback is due to the fact that compression algorithms work on the basis of spatial distances. However, considering the continuous nature of moving objects, both spatial and temporal factors should be taken into account in any compression technique.

In this paper, problems of existing compression techniques for moving object application are addressed. New spatio-temporal techniques have been proposed to overcome the mentioned problems. The experiments confirm the superiority of the proposed methods to the existing ones.

Clearly, obtained results depend on the chosen threshold. Choosing a proper threshold is not easy and is application-dependent. However, having a clear understanding of moving object behaviour helps

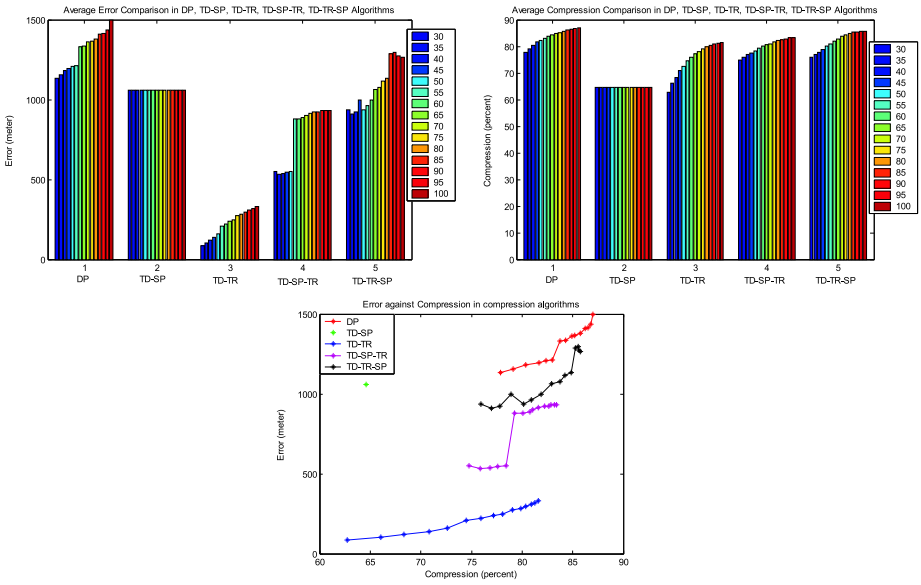

Figure 4: Comparison between conventional DP, TD-SP, TD-TR, TD-SP-TR and TD-TR-SP algorithms

in choosing an appropriate threshold.

In this paper, piecewise linear interpolation was used as the approximation technique. Considering the fact that other data such as speed and direction can directly be available, other interpolation techniques can be defined and be applied. The error notion used in existing, non-temporal techniques is not very suitable for moving object trajectories. This is an issue to be addressed in the future.

References

[1] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[2] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, second edition, 1990.

[3] J. Hershberger and J. Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. volume 1, pages 134–143, Charleston, South Carolina, USA, 3–7 August 1992. University of South Carolina, Humanities and Social Sciences Computing Lab.

[4] G. F. Jenks. Lines, computers, and human frailties. *Annuals of the Association of American Geographers*, 71(1):1–10, 1981.

[5] G. F. Jenks. Linear simplification: How far can we go? Paper presented to the Tenth Annual Meeting, Canadian Cartographic Association, 1985.

[6] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. pages 289–296, Silicon Valley, California, USA, 29 November – 2 December 2001. IEEE Computer Society.

[7] T. Lang. Rules for robot draughtsmen. *Geographical Magazine*, 42(1):50–51, 1969.

[8] Z. Li. An algorithm for compressing digital contour data. *The Cartographic Journal*, 25:143–146, December 1988.

[9] R. B. McMaster. Statistical analysis of mathematical measures of linear simplification. *The American Cartographer*, 13(2):103–116, 1986.

[10] R. B. McMaster. Automated line generalization. *Cartographica*, 24(2):74–111, 1987.

[11] W. R. Tobler. *Numerical Map Generalization*, chapter 8. Michigan Inter-University Community of Mathematical Geographers. University of Michagan, Ann Arbor, Michagan, USA, 1966.

[12] E. R. White. Assessment of line generalization algorithms using characteristic points. *The American Cartographer*, 12(1):17–27, 1985.