RJ 10007 (89096) January 24, 1996
Computer Science

Research Report

A MINIMUM DESCRIPTION LENGTH POLYGONAL APPROXIMATION METHOD

Saibal Banerjee
Wayne Niblack
Myron Flickner

IBM Research Division
650 Harry Road
San Jose, CA 95120

Research Division
Yorktown Heights, New York • San Jose, California • Zurich, Switzerland

A MINIMUM DESCRIPTION LENGTH POLYGONAL APPROXIMATION METHOD

Saibal Banerjee
Wayne Niblack
Myron Flickner

IBM Research Division
650 Harry Road
San Jose, CA 95120

ABSTRACT:

Given a sequence of points in two dimensions, obtained by sampling the perimeter of a planar region, it is often of interest to approximate this set by piecewise linear curves or polygons. In this paper we take a Minimum Description Length (MDL) approach to this problem. The algorithm presented yields the polygonal approximation with the (globally) minimum description length to the given sequence of points for fixed initial and final endpoints. The execution time and space requirements of our algorithm are both $O(n^2)$, where n is the number of points. Experimental results demonstrating the behavior of the optimal algorithm under changes in noise and precision levels are shown.

1. Introduction

Given a sequence of points in two dimensions, obtained by sampling the perimeter curve of a simple region, it is often of interest to approximate this sequence by a set of piecewise linear curves or by a polygon. Such approximations not only provide a compact description of the curve, but can also be used by efficient shape matching algorithms in OCR, industrial classification of machine parts, shape-based queries in image databases, etc. Since a compact description is sought, what could be more natural than the application of Minimum Description Length (MDL) principles to this problem?

The problem of piecewise linear or polygon approximations has a long history, and has received considerable attention in literature. Surveys are given in [1, 2, 3, 4], and, of particular relevance to this paper is the work of Itoh [5], who develops an MDL formulation of the problem. He provides a fast albeit suboptimal solution to the problem. Our MDL formulation is somewhat different from Itoh's, and we provide a fast and, within the model class we consider, optimal solution.

Section 2 starts out by giving an overview of the problem and the MDL criterion, and then derives a cost function for polygon approximation based on MDL. In Section 3 an optimization algorithm that finds the global minimum of this cost function is presented. It is shown that the time and space complexity of this algorithm are both quadratic in the number of points in the sequence. Experimental results and actual running times for the algorithm are shown in Section 4. Conclusions and open issues are given in Section 5.

2. Problem Domain

Given a simple, closed curve in two dimensions representing the perimeter of an object or region, a sequence C of two dimensional points arises by sampling the curve. We consider C to be an ordered list of n two dimensional coordinates (x_i, y_i). Sometimes, we shall use the notation p_i to represent the ith point in the sequence. That is,

$$C = \{p_i = (x_i, y_i) \mid 0 \leq i < n\}$$

For example, C can be given by the sequence of points shown in Figure 1(a). A polygonal or linear piecewise approximation to such a sequence C of data points is a sequence U of k points in two dimensions,

$$U = \{w_i \in \mathcal{R}^2 \mid 0 \leq i < k\}$$

such that the lines joining adjacent points in U approximates the point sequence C. We shall often refer to the members of U as line endpoints. Further, we shall assume that we are dealing with polygonal approximations to data points sampled along a closed curve.

Both Figures 1(b) and (c) represent polygonal approximations to the data in Figure 1(a). To select among the different approximations, a criterion is needed, such as small maximum error, simplicity, or, typically, some combination of both. We derive such a criterion, or cost function, using the MDL principle.

To simplify the development, we make two common assumptions:

Assumption 1. *The endpoints are a subsequence of the given data points. That is, $Z \subseteq C$.*

Assumption 2. *The starting point ($\equiv$ ending point) of the polygonal approximation is specified. Furthermore, this point is the first point in the data point sequence.*

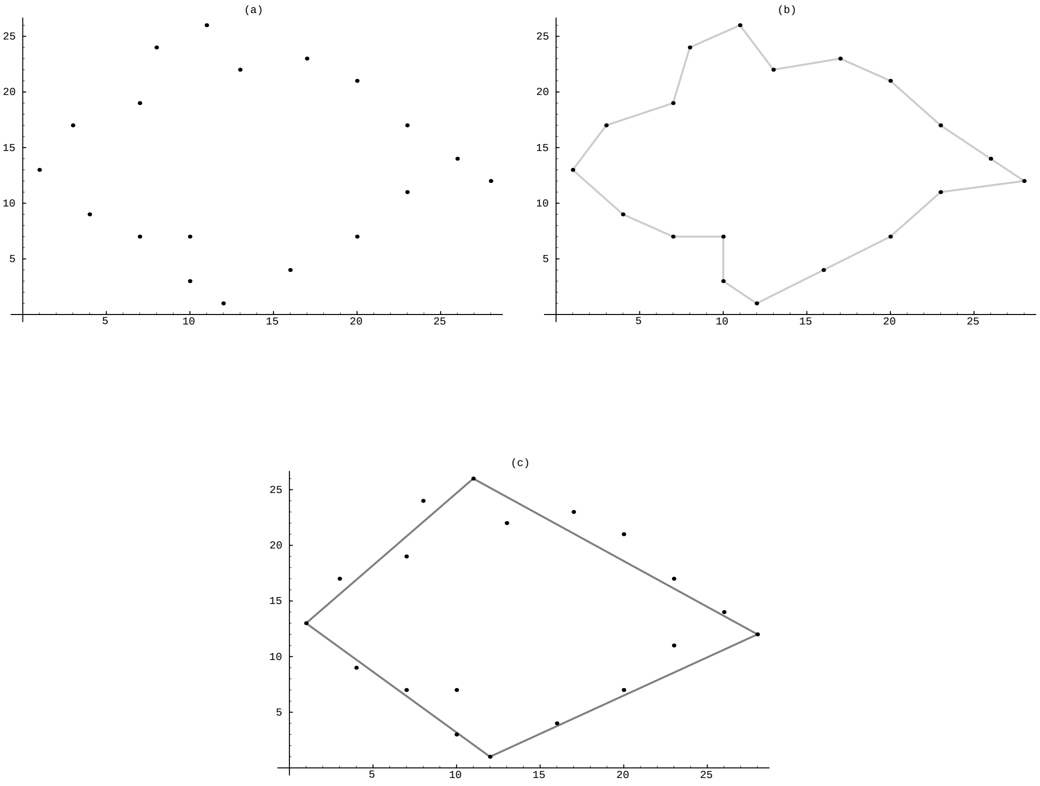

Figure 1: (a) A test sequence of points. (b) Ordering of the sequence. (c) A possible polygonal approximation to the sequence.

These assumptions, and methods to avoid them, are re-visited in in Section 5.

In the next section, we provide an introduction to the MDL criterion, and in the sections that follow, use it to construct a cost function for polygonal approximation.

2.1. The MDL Criterion

The MDL criterion provides a general framework for selecting a model from (or for) a set of data. The general problem may be stated as follows. Given a set of data $U = \{u_1, u_2, \ldots, u_n\}$ and a class of models $\mathcal{M} = \{M_1, M_2, \ldots\}$, each capable of describing U, select the best model. The common situation is when we are dealing with probabilistic models, in which case when we say a model M_i is "capable of describing U" we mean we can write an expression for $p(U|M_i)$. The MDL criterion has its origins in coding and information theory, and, in general terms, states that the best model is the one allowing the shortest encoding of the data.

It is often helpful to view the MDL criterion in the context of a communication system between two parties—a Sender and a Receiver, as depicted in Figure 2. Both the Sender and the Receiver use their individual code books. These code books must be identical in order to transmit data from Sender to Receiver. Thus the process consist of:

1. An agreement phase in which the Sender and the Receiver decide on what protocol to follow in communicating the data and how to represent the data. It is at this stage that the sequential order in which the various pieces of information are communicated and

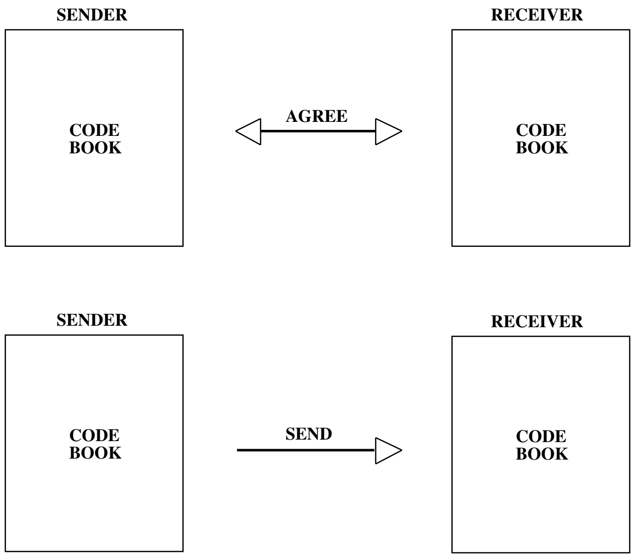

Figure 2: MDL viewed as a Communication Process

their semantics are agreed upon. It is also at this stage that the model class(es) is (are) agreed to, and the common codebooks, or rules for constructing them, are defined.

2. An encoding, transmission, and decoding phase in which the Sender is given data (the Receiver is incapable of obtaining data except from the Sender), selects a model with which to encode the data, and as per the agreement, encodes and sends it to the Receiver such that the latter can reconstruct it *exactly*.

It is the Sender's task to define a codebook to allow efficient, ideally minimal-length, encoding. To keep the message self-contained, the Sender must also include the codebook in the message. A given model M acts like the codebook, or, more precisely, provides instructions for generating a codebook. Thus in the MDL framework, one imagines sending the model, and then the data encoded using codebook determined by the model. The total message codelength is $L(U, M)$, the length for the data and the model. This, by Shannon, is:

$$L(U, M) = -\log p(U, M) \tag{1}$$

(all logs are assumed to be of base 2), which we may further write as

$$L(U, M) = -\log p(U, M) = -\log p(U|M) - \log p(M) \tag{2}$$

The first term on the right is the codelength of U encoded using model M, the second is the codelength of M. The MDL principle states that the best model is the one for which this combined expression is minimum.

Given Equation 2, different choices are possible as to how a model is specified and encoded. Suppose for example that the data U represents the y values of coordinates $\{0, 1, 2, \ldots, n\}$, and we seek to model this data with a polynomial curve, and we are free to choose both the order and coefficients of the polynomial. Further, we will model the data as a polynomial curve plus noise, allowing some deviation from an exact interpolating polynomial (which is not only uninteresting, but also a subcase of the noise case). We must still exactly describe each data point, but may do so by describing it as a value of the polynomial plus a deviation.

The simplest MDL approach is to consider encoding the data in a message that includes (a) the polynomial order and coefficients, and (b) the residual error or deviation at each point from the specified polynomial. It is up to the Sender to find the polynomial/deviation set giving the shortest combined codelength, but this is possible by a combination of exhaustive search through different degree polynomials combined with least squares fitting. Such a method is referred to by Rissanen [6] as two part coding since the message explicitly includes a part to fully specify the model, followed by a part to describe the data. (Other forms of MDL, such as predictive MDL and mixture density encoding do not explicitly encode the model parameters and hence are, in fact, more efficient methods.) As an important detail, Rissanen has also shown that the encoding of real-valued model parameters (the coefficients in this case) derived from n data points can be done with an effective codelength of (asymptotically as n grows large) $\frac{1}{2}\log n$ bits per parameter. His derivation is based on truncating the real valued parameters to an optimal precision so that their codelength is small (tending to low precision and short codelengths), but the codelength of the data encoded using the truncated parameters is also small (tending to large precision and long codelength for the parameters). We refer to this term as the parameters' codelength due to "precision optimization". When it is included, we get the most common form of MDL and the one we will use in this paper, in which the total codelength is written as:

$$L(U, M_k) = -\log p(U|M_k) + \frac{K}{2}\log n \tag{3}$$

where K is the number of parameters (such as the number of coefficents in the polynomial) required by model k. The last term is often called the "penalty term" since it penalizes complex (i.e. many parameter) models.

2.2. The Problem Formulation

An observation is a sequence of data points in two dimensions. In order to encode such an observation the Sender and Receiver agree on the following:

1. A subsequence of the data points will be sent as endpoints of a polygon.
2. The rest of the points will be encoded as "deviations" of points sampled from the sides of this polygon.

In order to satisfy the MDL criterion, the Sender must send a polygon such that the overall codelength of the polygon plus the deviations is a minimum. Such a polygon is the MDL polygonal approximation to the given sequence of data points.

Thus, an agreement must be reached on two things: (i) how to code a polygon, and (ii) how to code the sequence of points that the sides of the polygon "explains" as deviations. Since the points are given in sequential order it is natural to have the following agreement:

1. Every point in C is explained by one line of the polygon.
2. The points in C are explained in sequential order. i.e. if $p_i = w_m$ and $p_j = w_{m+1}$, where $j > i$ and the w's represent the polygon vertex sequence, then the points $p_{i+1}, \ldots, p_{j-1}$ must be encoded as deviations from the line joining w_m and w_{m+1}.

The reader is reminded that the encoding of the deviations must be such that the Receiver can reconstruct exactly each one of these data points. Note also that the data to be encoded are given to some fixed, finite precision – say the precision of the measuring device or the precision of the numeric representation in the Sender and Receiver computers. Thus we need not be concerned with encoding infinite precision but can agree on some specific precision. Further, we can then imagine scaling the data so that this precision is unity, and this is what we shall do. See also the comments in Section 5.

In the next two subsections we explain how a polygon is modeled, and, given a polygon, how the deviations are modeled.

2.3. The Polygon Model

In order to specify a polygon, one must first state the number of sides it has, or equivalently the number of endpoints. Thus, one must code an integer. There are numerous ways to code integers universally, one of the most efficient being the Elias code [7]. The Elias code for the integer k requires approximately $\log^* k$ bits.[1]

Next, one must code the endpoints themselves. If the number of sides of the polygon is k, this requires specifying a sequence of k coordinate (x, y) values, which, because of the scaling mentioned above, we can consider as integers. Given the data, the Sender could first send the coordinates of the smallest isothetic rectangle bounding the observation, i.e. the four values x_{max}, x_{min}, y_{max}, and y_{min}. Given these four values, and assuming a uniform distribution, one can code any x-coordinate inside the rectangle with $\log(x_{max} - x_{min})$ bits and any y-coordinate with $\log(y_{max} - y_{min})$ bits. Thus the codelength for an endpoint or equivalently for a side of the polygon is $\log(x_{max} - x_{min})(y_{max} - y_{min}) = \log A$ bits, where A is the area of the bounding isothetic rectangle. Note that the four coordinate values of the bounding rectangle have to be sent once for a set of data. It can be regarded as a constant piece of code, and need not enter explicitly into the MDL cost function comparing different polygonal models as described below.

Once the number k of sides, and the k endpoints have been received by the Receiver, she can construct the polygon (i.e. polygonal approximation). In the next subsection, we show how the rest of the data points are coded as being noisy samples taken from the sides of this polygon.

2.4. The Sample and Noise Model

To encode the points explained by a line, there are three things one has to do:

1. Code the number of points explained by the line. This may be done in various ways – assuming a uniform distribution in the range $[0, n]$; assuming a uniform distribution in the range $[0, n']$, where $n' = (n-$the number of points sent in previous segments); or using the universal code for integers, i.e. if there are m points, the codelength is $\log^* m$ bits.
2. Code the deviation of each point from the line, such that the exact location of each point can be recovered by the Receiver. It is here that some probability models are required—conditional probabilities that specify a spatial deviation of a point from a given line. The codelength to specify a particular point is then the negative logarithm of this spatial deviation probability.

[1] $\log^* n$ is defined as $\log n + \log\log n + \cdots$ over all positive terms.

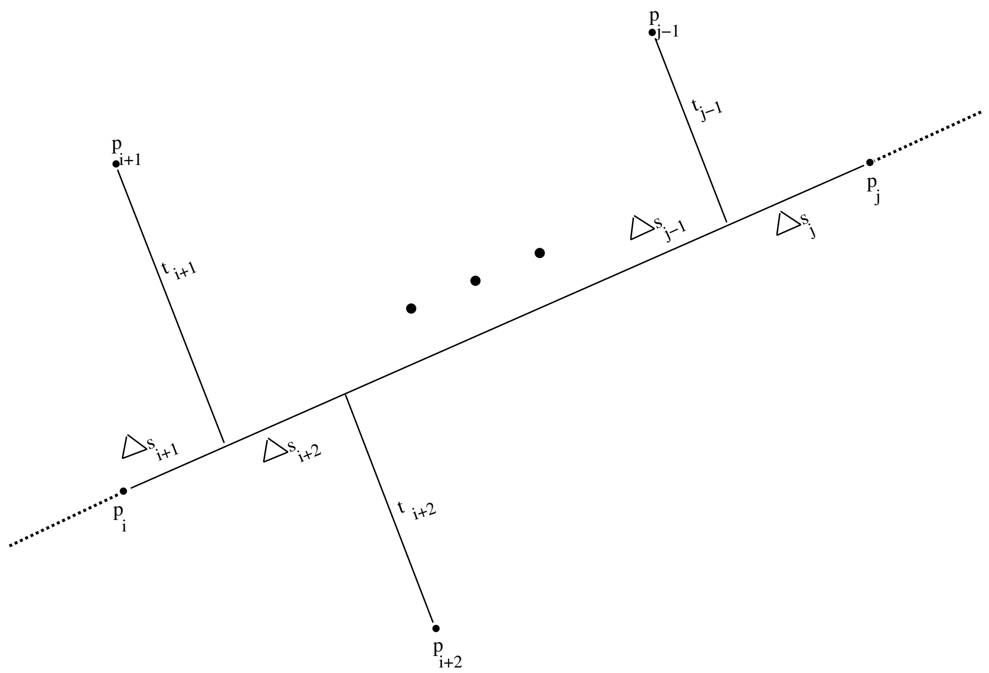

Figure 3: The line L

3. Once a probability model (i.e. a distribution) is specified, there could be one or more parameters associated with the distribution. For example, a one-dimensional Gaussian distribution is parametrized by two variables—the mean and the standard deviation. The Sender also needs to send the values of these parameters to the Receiver. There are two possibilities here:
   (a) The same parameter values are associated with every line. In this case the Sender has to send this once, and therefore it can be regarded as a constant piece of code, independent of the number of lines. Hence the codelength required for this need not enter explicitly into the MDL cost function.
   (b) Each line has a separate set of parameter values. We shall assume this model. In this case the Sender is required to send these values once for every line, and our expectation is that this increase in codelength will be offset by a corresponding decrease in codelength for the individual points since they can be encoded with a more efficient code, adapted to each line. In any case, if m is the number of points explained by a line which requires such parameters, then the optimal precision codelength required to send each parameter is, as in Equation 3, approximately $\frac{1}{2}\log m$. (See [8] for details on optimal precision codelengths.)

The probability models that we shall assume are defined next.

Consider the line L passing through the data points p_i and p_j, where $j > i$, as in Figure 3. Consider the points $p_{i+1}, \ldots, p_{j-1}$ that the line L has to explain. Drop perpendiculars from the points $p_{i+1}, \ldots, p_{j-1}$ onto L. One of the ways that L can explain these points is by specifying the lateral and perpendicular distances of each of these points along L – i.e. code the intersampling distance Δs between each point, and the signed, perpendicular distance t of each point perpendicular to L.

We will assume that both the Δs's and the t's are independently distributed with a Gaussian distribution. We assume furthermore that the individual Δs's and t's for the points explained by the line L are all independent random variables. With these assumptions, in order to minimize the codelength for sending the individual points $p_{i+1}, \ldots, p_{j-1}$ we need to send the maximum-likelihood (ML) values for the means and the standard deviations. However, as the line L should pass near the center of the data points that it explains, we will assume that the perpendicular distances t have zero mean about L. Thus, if $\mathbf{\Delta s}$ and $\mathbf{t}$ represent the random variables for the lateral interpoint distances and the perpendicular distances respectively, then $\mathbf{\Delta s} \sim \mathbf{N}(\hat{\mu}_{\Delta s},\, \hat{\sigma}_{\Delta s})$ and $\mathbf{t} \sim \mathbf{N}(0,\, \hat{\sigma}_t)$, where $\hat{\mu}_{\Delta s}$ and $\hat{\sigma}_{\Delta s}$ are the ML mean and ML standard deviation of the Δs's, and $\hat{\sigma}_t$ is the ML standard deviation of the t's. The Sender must therefore send these three values to the Receiver for each line, before encoding the points explained by this line.

Consider the calculation of the ML mean $\hat{\mu}_{\Delta s}$ for the line L, with data points p_i and p_j $(n \geq j > i \geq 0)$ as its endpoints. Let $m \geq 0$ denote the number of data points explained by line L, i.e. $m = j - i - 1$. Then $\hat{\mu}_{\Delta s}$ for the m values of Δs is given by (see for example [9]):

$$\hat{\mu}_{\Delta s} = \frac{\sum_{r=i+1}^{j-1} \Delta s_r}{m} \approx \frac{\sum_{r=i+1}^{j} \Delta s_r}{m+1} = \frac{D_{ij}}{m+1} \tag{4}$$

where D_{ij} is the Euclidean distance between the points p_i and p_j, and the approximation results because we have included the last Δs even though it is not encoded (because p_j is already known).

Thus, the knowledge of the points p_i and p_j which are sent before as endpoints of the polygon, and the number of points m explained by the line L, are sufficient for the Receiver to deduce the value of $\hat{\mu}_{\Delta s}$. Thus the Receiver requires, and the Sender need send only two of the three parameter values, namely the ML standard deviations $\hat{\sigma}_{\Delta s}$ and $\hat{\sigma}_t$.

Now consider the calculation of the ML standard deviations $\hat{\sigma}_{\Delta s}$ and $\hat{\sigma}_t$ for the line L. These are given as follows ([9]):

$$\hat{\sigma}^2_{\Delta s} = \frac{\sum_{r=i+1}^{j-1} (\Delta s_r - \hat{\mu}_{\Delta s})^2}{m} = \frac{\sum_{r=i+1}^{j-1} \Delta s_r^2}{m} - \hat{\mu}^2_{\Delta s} \tag{5}$$

$$\hat{\sigma}_t^2 = \frac{\sum_{r=i+1}^{j-1} t_r^2}{m} \tag{6}$$

Thus, using Rissanen's precision optimization encoding method, codelengths of $\frac{1}{2}\log m$ would be required for encoding each of the parameters $\hat{\sigma}_{\Delta s}$ and $\hat{\sigma}_t$ for L.

Next we consider the codelengths required to send the individual values of the Δs's and the t's for the line L using the Gaussian distribution assumption with the ML standard deviations. By the Gaussian assumption, the probability distribution of Δs_r for $i < r \leq j$, with the ML parameters, is given by:

$$\frac{1}{\sqrt{2\pi}\hat{\sigma}_{\Delta s}} \exp\left[-\frac{(\Delta s_r - \hat{\mu}_{\Delta s})^2}{2(\hat{\sigma}_{\Delta s})^2}\right]$$

By the independence assumption, the joint distribution for all the Δ_s values along the line segment is thus equal to:

$$p(\Delta s_{i+1}, \Delta s_{i+2}, \ldots, \Delta s_{j-1} | \hat{\mu}_{\Delta s}, \hat{\sigma}_{\Delta s}) = \prod_{r=i+1}^{j-1} \frac{1}{\sqrt{2\pi}\hat{\sigma}_{\Delta s}} \exp\left[-\frac{(\Delta s_r - \hat{\mu}_{\Delta s})^2}{2(\hat{\sigma}_{\Delta s})^2}\right]$$

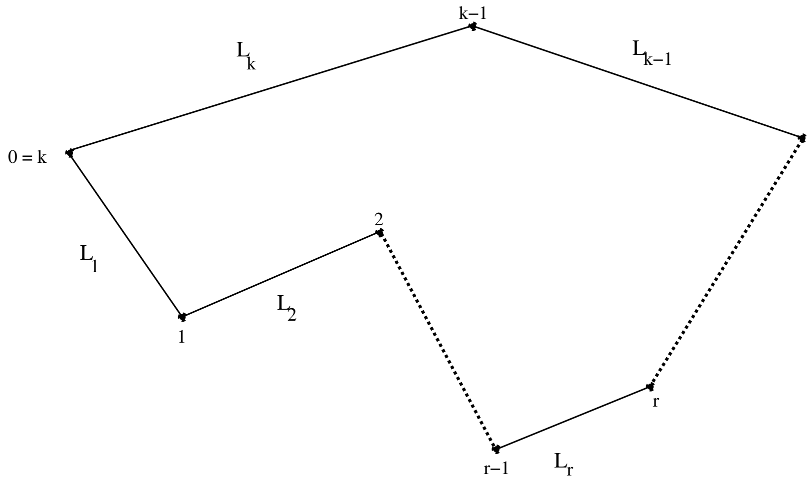

Figure 4: A k sided polygon

$$= \left(\frac{1}{\sqrt{2\pi}\hat{\sigma}_{\Delta s}}\right)^m \exp\left[\frac{-\sum_{r=i+1}^{j-1}(\Delta s_r - \hat{\mu}_{\Delta s})^2}{2(\hat{\sigma}_{\Delta s})^2}\right]$$
$$= \left(\frac{1}{\sqrt{2\pi}\hat{\sigma}_{\Delta s}}\right)^m e^{-\frac{m}{2}}$$

where the last step follows from the definition of the ML standard deviation in Equation 2. Now by Shannon theory, the codelength is given by

$$-\log\left[\left(\frac{1}{\sqrt{2\pi}\hat{\sigma}_{\Delta s}}\right)^m e^{-\frac{m}{2}}\right] = \frac{m}{2}\log 2\pi e\hat{\sigma}_{\Delta s}^2 \tag{7}$$

By a similar reasoning, the codelength for the t_r's $(i < r < j)$ is given by

$$\frac{m}{2}\log 2\pi e\hat{\sigma}_t^2 \tag{8}$$

2.5. The Problem Statement

Consider a k sided polygon with lines $L_1 \ldots L_k$ as shown in Figure 4. Let A be the area of the smallest isothetic rectangle bounding the data points. Let $m_r \geq 0$ denote the number of data points explained by line L_r for $1 \leq r \leq k$. Also, let $\hat{\sigma}_{\Delta s}^r$ and $\hat{\sigma}_t^r$ be the lateral interpoint ML standard deviation and the perpendicular ML standard deviation respectively for the line L_r $(1 \leq r \leq k)$.

Thus, by the previous discussion, the total codelength (except for the constant pieces of code) required to encode all the data points is given by

$$\log^*(k) + k \times \log A +$$
$$\sum_{r=1}^{k}\left(\log^* m_r + \frac{1}{2}\log(m_r) + \frac{1}{2}\log m_r + \frac{m_r}{2}\log 2\pi e(\hat{\sigma}_{\Delta s}^r)^2 + \frac{m_r}{2}\log 2\pi e(\hat{\sigma}_t^r)^2\right)$$

Next we make certain approximations to the above codelength. Note that as $k \times \log A \gg \log^*(k)$, we can neglect the term $\log^*(k)$ in the overall codelength. This will allow us to define the codelength H_r for the line L_r individually as follows:

$$\begin{aligned} H_r &= \log A + \log^* m_r + \frac{1}{2}\log(m_r) + \frac{1}{2}\log m_r + \\ &\quad \frac{m_r}{2}\log 2\pi e(\hat{\sigma}^r_{\Delta s})^2 + \frac{m_r}{2}\log 2\pi e(\hat{\sigma}^r_t)^2 \end{aligned} \tag{9}$$

The overall codelength is now simply the sum $\sum_{r=1}^{k} H_r$. By a slight abuse of notation, let H_{ij} (for $0 \le i < j \le n$) denote the codelength for the line starting from data point p_i and ending at data point p_j. By Assumption 2, these are the only possible codelengths one need consider. Then the we can state the problem we seek to solve as:

The Problem: Find the subsequence (see Figure 4) $0 \equiv i_0 < i_1 < \cdots < i_k \equiv n$ such that $\sum_{r=1}^{k} H_{i_{r-1} i_r}$ is minimized.

3. Problem Solution

In this section we show how one can solve the MDL polygonal approximation problem. The solution can be broken up into three stages:

1. The first stage consists of evaluating the $\log A$ term. This can be easily accomplished during the time that the data point sequence is read in, by updating the four variables x_{max}, x_{min}, y_{max} and y_{min} after each point is read in, and finally performing the requisite arithmetic operation.
2. In Section 3.1 the codelengths H_{ij} are evaluated for all $0 \le i < j \le n$. It is shown how one can calculate the codelengths $H_{i\,i+1} \ldots H_{i\,n}$ of all lines starting at the data point p_i iteratively with a constant time required per iteration.
3. In Section 3.2 we show how one can find the optimal solution to our problem using these codelength values (i.e. the H's) and a standard minimum path searching algorithm.

3.1. Codelength evaluation for all lines

Consider the line L_{ij} starting from the data point p_i and ending at the data point p_j $(j > i)$ as in Figure 3. We shall rewrite its codelength H_{ij} in a way such that an iterative procedure to find the codelengths of all lines starting at p_i is possible.

Since the number of points that L_{ij} explains is $m_{ij} \equiv (j - i - 1)$, from Equation 9 we get,

$$\begin{aligned} H_{ij} &= \log A + \log^* m_{ij} + \frac{1}{2}\log m_{ij} + \frac{1}{2}\log m_{ij} + \\ &\quad \frac{m_{ij}}{2}\log 2\pi e\hat{\sigma}^2_{\Delta s} + \frac{m_{ij}}{2}\log 2\pi e\hat{\sigma}^2_t \end{aligned} \tag{10}$$

where the parameters $\hat{\sigma}_{\Delta s}$ and $\hat{\sigma}_t$ are given by Equations 5 and 6. That is,

$$\hat{\sigma}^2_{\Delta s} = \frac{\sum_{r=i+1}^{j-1} \Delta s_r^2}{m_{ij}} - ((\hat{\mu}_{\Delta s})_{ij})^2 \tag{11}$$

$$\hat{\sigma}^2_t = \frac{\sum_{r=i+1}^{j-1} t_r^2}{m_{ij}} \tag{12}$$

where $(\hat{\mu}_{\Delta s})_{ij}$ is estimated as

$$(\hat{\mu}_{\Delta s})_{ij} = \frac{D_{ij}}{m_{ij}+1} \tag{13}$$

and D_{ij} is the Euclidean distance between p_i and p_j. Define two new variables for the sum-of-squares terms that occur in Equations 11 and 12, namely,

$$S_{ij} = \sum_{r=i+1}^{j} \Delta s_r^2 \tag{14}$$

$$T_{ij} = \sum_{r=i+1}^{j-1} t_r^2 \tag{15}$$

Using Definitions 14 and 15, and Equations 11 and 12, we can rewrite Equation 10 as follows:

$$\begin{aligned} H_{ij} = {} & \log A + \log^* m_{ij} + \frac{1}{2}\log m_{ij} + \frac{1}{2}\log m_{ij} + \\ & \frac{m_{ij}}{2}\log 2\pi e \left[\frac{S_{ij}}{m_{ij}} - ((\hat{\mu}_{\Delta s})_{ij})^2\right] + \frac{m_{ij}}{2}\log 2\pi e \left[\frac{T_{ij}}{m_{ij}}\right] \end{aligned} \tag{16}$$

It should now be obvious that if we can find iterative methods to calculate the S's and the T's, we are done. To this end, let (x_r, y_r) denote the coordinates of data point p_r for $0 \le r \le n$. We shall now evaluate the two sum-of-squares terms which occur in the calculation of H_{ij} in Equation 16.

Let θ be the (anti-clockwise) angle made by the line L_{ij} with the positive x-axis. Then for $i < r \le j$,

$$\Delta s_r = (x_r - x_{r-1})\cos\theta + (y_r - y_{r-1})\sin\theta \tag{17}$$

And for $i < r < j$,

$$t_r = (y_r - y_i)\cos\theta - (x_r - x_i)\sin\theta \tag{18}$$

Equations 17 and 18 are the fundamental equations which yield the results we seek. Now note that,

$$\cos\theta = \frac{x_j - x_i}{D_{ij}} \tag{19}$$

$$\sin\theta = \frac{y_j - y_i}{D_{ij}} \tag{20}$$

Next, we consider the iterative evaluation of the S's. Define the extra variables:

$$S_{ij}^{XX} = \sum_{r=i+1}^{j} (x_r - x_{r-1})^2 \tag{21}$$

$$S_{ij}^{YY} = \sum_{r=i+1}^{j} (y_r - y_{r-1})^2 \tag{22}$$

$$S_{ij}^{XY} = \sum_{r=i+1}^{j} (x_r - x_{r-1})(y_r - y_{r-1}) \tag{23}$$

Squaring both sides of Equation 17 and adding over r ($i < r \leq j$) we get from Definition 14 that,

$$\begin{aligned} S_{ij} &= \cos^2\theta \sum_{r=i+1}^{j} (x_r - x_{r-1})^2 + \sin^2\theta \sum_{r=i+1}^{j} (y_r - y_{r-1})^2 + \\ & \quad 2\sin\theta\cos\theta \sum_{r=i+1}^{j} (x_r - x_{r-1})(y_r - y_{r-1}) \\ &= \frac{(x_j - x_i)^2}{D_{ij}^2} S_{ij}^{XX} + \frac{(y_j - y_i)^2}{D_{ij}^2} S_{ij}^{YY} + 2\frac{(x_j - x_i)(y_j - y_i)}{D_{ij}^2} S_{ij}^{XY} \end{aligned} \tag{24}$$

The updating of S^{XX}, S^{YY}, and S^{XY} is immediate from Definitions 21, 22, and 23 respectively. For example,

$$S_{i\,j+1}^{XX} = S_{ij}^{XX} + (x_{j+1} - x_j)^2 \tag{25}$$

This in turn allows us to update the sum-of-squares S via Equation 24.

A similar treatment can be carried out for the T's, defining, for example:

$$T_{ij}^{XX} = \sum_{r=i+1}^{j-1} (x_r - x_i)^2 \tag{26}$$

which can be updated incrementally, so that we have a fast way to compute T in Equation 15. Once the S and the T has been updated, we can update H via Equation 16.

The result is that, by keeping the six extra variables S^{XX}, S^{YY}, S^{XY}, T^{XX}, T^{YY}, and T^{XY}, we can update the value of H from H_{ij} to $H_{i\,j+1}$ using a constant number of arithmetic operations. We use this updating technique to find in sequence each row of the matrix $\mathcal{H}$ defined as

$$\mathcal{H} = \begin{bmatrix} H_{01} & H_{02} & \cdots & H_{0n} \\ & H_{12} & \cdots & H_{1n} \\ & & \vdots & \\ & & & H_{n-1\,n} \end{bmatrix} \tag{27}$$

With $\mathcal{H}$, we are ready for the next step of the algorithm, the optimization.

3.1.1. Time and Space Complexities

For this part of the algorithm we obviously need $O(n^2)$ space in order to store $\mathcal{H}$, and evidently we do not use more space than this in order to compute it.

As we have shown, it takes a constant time to update the entries from one column to the next along a particular row of $\mathcal{H}$. Since the ith row of $\mathcal{H}$ has $n - i$ entries, it requires $O(n - i)$ time to fill up the ith row. It therefore requires $\sum_{i=0}^{n-1} O(n - i) = O(n^2)$ time to compute the entire matrix $\mathcal{H}$.

3.2. The Optimization Step

Using the entries in $\mathcal{H}$ defined above, we next find the subsequence $0 \equiv i_0 < i_1 < \cdots < i_k \equiv n$ which minimizes the sum $\sum_{r=1}^{k} H_{i_{r-1} i_r}$.

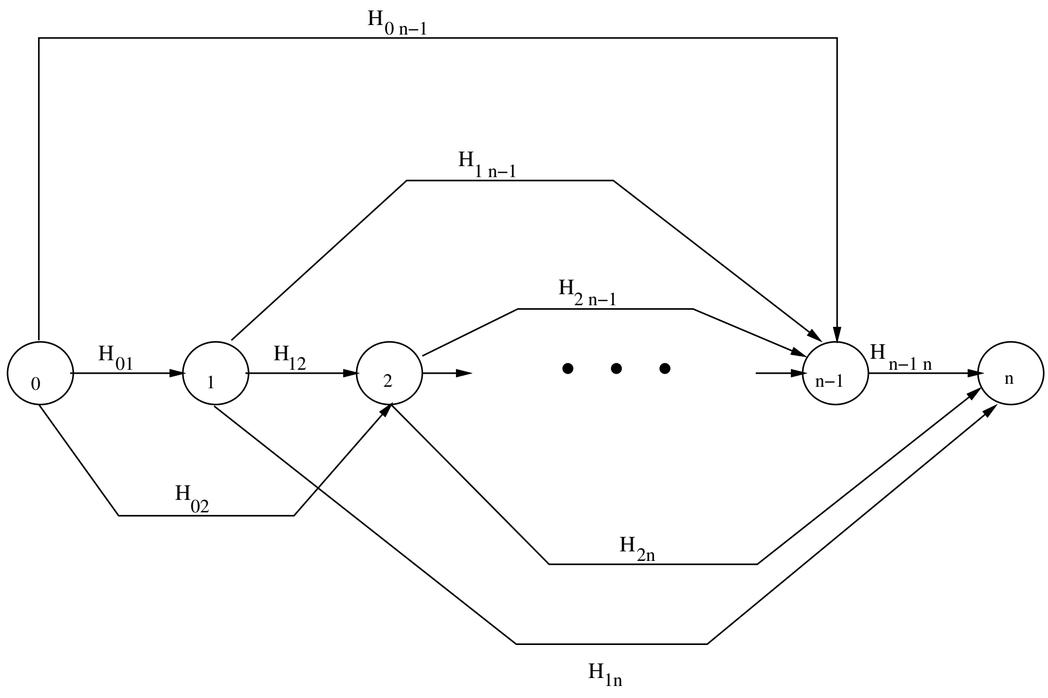

Figure 5: The graph $\mathcal{G}$

There are numerous ways to think about this optimization problem. However, the simplest and most graphical seems to be in graph theoretic terms.

Define a directed graph $\mathcal{G}$ (see Figure 5 on $n+1$ vertices labeled $0, \cdots, n$. Let the forward arcs i.e. arcs from vertex i to vertex j for $i < j$, be weighted with weight H_{ij}. All backward arcs and the arc from vertex 0 to n are of infinite weight.

Now consider any directed path P from vertex 0 to vertex n in $\mathcal{G}$. P is a subsequence $0 \equiv i_0 < i_1 < \cdots < i_k \equiv n$ of vertices, corresponding to the arcs $\{0\, i_1\}, \cdots \{i_{k-1}\, n\}$ in $\mathcal{G}$ that it traverses. (Traversal of infinitely weighted arcs are disallowed.) If the weight of a path is defined as the sum of the weights on the arcs it traverses, then the weight of P is given by $\sum_{r=1}^{k} H_{i_{r-1} i_r}$. It immediately follows that our problem reduces to that of finding the shortest path from vertex 0 to vertex n in $\mathcal{G}$.

The shortest path problem is a widely studied problem and efficient algorithms exist for its solution. One of the popular algorithms to solve this problem is a greedy algorithm known as Dykstra's algorithm. The reader is referred to [10] for details. The essence of this algorithm is to start with a table $A[i, j]$ giving the distance between every pair of input data points. Also start with an array B of size $n-1$ of the distance from the starting point to each of the other $n-1$ points. The shortest path between the starting point and ending point (in fact, between the starting point and any other point) is found by a process that repeatedly performs the following steps:

- Find any remaining point, say q, whose distance from the starting point is equal to the minimum distance of any point to the starting point. (All points other than the starting point are originally "remaining" points.)
- Update the distance of the path to any point r if it is shorter to go to r by way of q.
- Remove q from the list of remaining points.

We apply Dykstra's algorithm and solve our optimization problem.

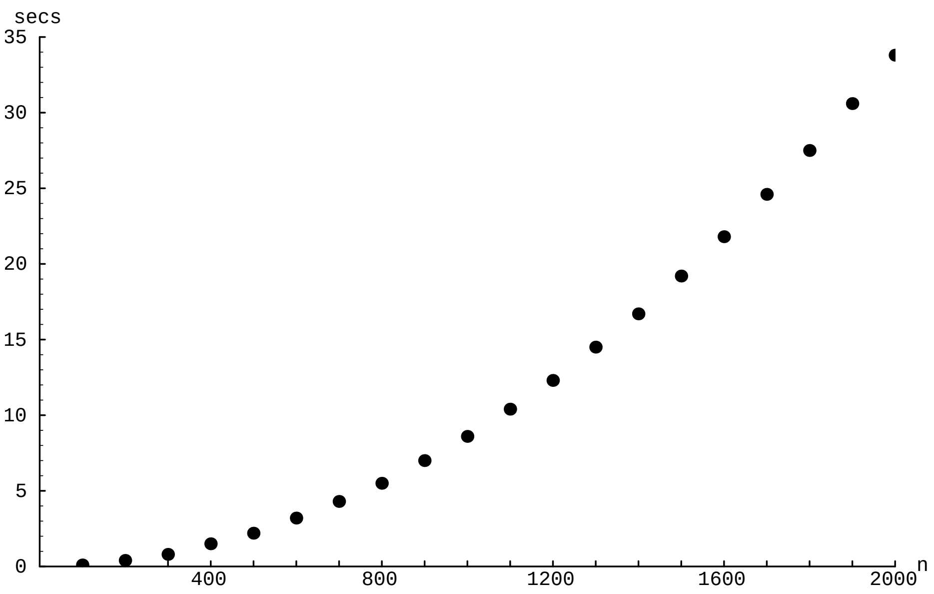

Figure 6: Graph of actual execution times

3.2.1. Time and Space Complexities We note that the graph $\mathcal{G}$ has $n+1$ vertices and $O(n^2)$ arcs. Thus, Dykstra's algorithm requires $O(n^2)$ time and space (see [10]). Therefore both the codelength evaluation step and the optimization step have $O(n^2)$ time and space bounds, and the overall time and space complexity of our algorithm are quadratic in the number of points. The actual running time of this algorithm is remarkably fast, i.e. there are very small constants in the Big-Oh term for the execution time. Run times are shown in Figure 6 where we have plotted the physical running time (in seconds) of our algorithm (including I/O) on an IBM RS6000 as a function of the number of points n. In all practical cases that we have seen, the number of points never exceeds 800, lying mostly between 200 and 600. From the graph in Figure 6, this means that the execution time of this algorithm is less than 5.5 seconds for all practical purposes. Considering the optimality criterion satisfied by the algorithm, such a result is notable.

4. Experiments

Figure 7 shows results on two sample, real world figures. The results are quite good, with nice Gestaltish properties. This is true even though the input may not (in fact, certainly does not) conform to some of the assumptions we made.

An additional result, showing the effects of noise in the input points, is given in Figure 8. We selected a Chinese character, whose sampled point sequence is shown in the top left corner, and applied synthetic increasing spatial noise to the figures, as depicted on the left, from top to bottom. The 2nd figure from top (on left) is formed by moving a dot in the top left figure 1 pixel to the left, right, up or down with equal probability. The 3rd and 4th figures (on left) are formed similarly by translations of 2 and 3 pixels respectively. Needless to say, this kind of spatial noise is rather drastic, and unlikely to occur naturally. This type of noise is not only non-Gaussian, but it tends to form locally self-intersecting paths in the sequence—a phenomenon which could never occur in sampling the perimeter of a closed curve. Still, the algorithm shows remarkable

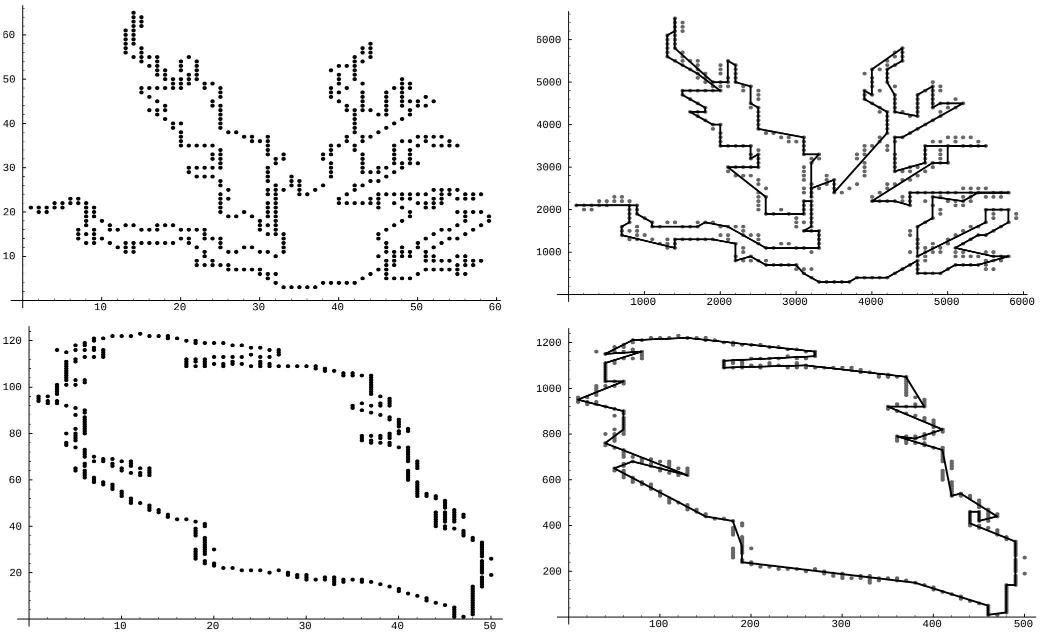

Figure 7: Top left: Lake Lewisville near Dallas, TX. Top right: MDL Polygon (overlaid on points) (scale 100). Bottom left: Right lung figure. Bottom right: MDL Polygon (overlaid on points) (scale 10).

noise filtering properties. This is demonstrated by the corresponding figures on the right, with the MDL Polygons overlaid on the point sequence.

In general, the higher the noise, the fewer the number of sides of the polygon there will be. This is because noise acts as a constraint in the MDL algorithm, which has to deduce the parameters of the noise from the data. If there is very little noise deduced, then a more exact fit is required, because otherwise the codelength due to noise is bound to blow up. This tends to increase the number of sides of the polygon. There is a tradeoff between "noise codelength" and "line codelength", but the algorithm will generally favor "noise codelength" when the data is less noisy.

5. Comments, Future Research and Conclusions

We list here some comments as well as ways the work can be carried further:

1. As is standard with applications of MDL, we have presented a method which does not require parameters or thresholds. No maximum allowed error or maximum number of points is specified. In place of such parameters, MDL requires that a model class (or classes) be specified, and our model class was that of straight line segments with Gaussian distributed Δs and t values. Within this class, MDL found the best set of segments, and the best values for the corresponding mean and variance parameters. This parameter free property of MDL is often very appealing.

2. In the development of our algorithm we assumed that the precision of the input data was unity. The effect of this assumption is felt primarily in the use of the models for the distributions of Δs and t. These were modeled as Gaussians, and if, using the quantization to unity, the values of Δs and t are not Gaussian distributed, we may obtain poor results. Practically, this will typically show up when taking the log of the standard deviation which has been computed to be a number less than one, giving a negative codelength. A simple solution is to scale the data so that the standard deviations of these distributions is, say, on the order of 8 to 10 – i.e. this defines a "natural scale" for the data.

3. Assumption 2, made in most literature, is neither visually nor intellectually satisfying. It leads to obviously suboptimal results, and to lines which look skewed because the endpoints are themselves noisy datapoints. A more satisfactory assumption would be to allow the endpoints to be near the data points. (Allowing them to be anywhere leads to undesirable results, such as for two nearly parallel lines with a join point far from the input points.) This is possible, for example, by including in the optimization search every point that is within some neighborhood of any input point. For neighborhoods of size h, the search would then be $O(hn^2)$.

4. One can directly extend this work to using not only lines, but also circular arcs for fitting point sequences. The method remains much the same, with two directed arcs, one for a line fit, and the other for a circular arc fit, between every two ordered vertices in the graph $\mathcal{G}$.

5. The method, like many other methods in literature, is starting point dependent. In order to make it independent of starting point, one can repeat it with each data point as the starting point and determine the overall minimum. Considering that the problem amounts

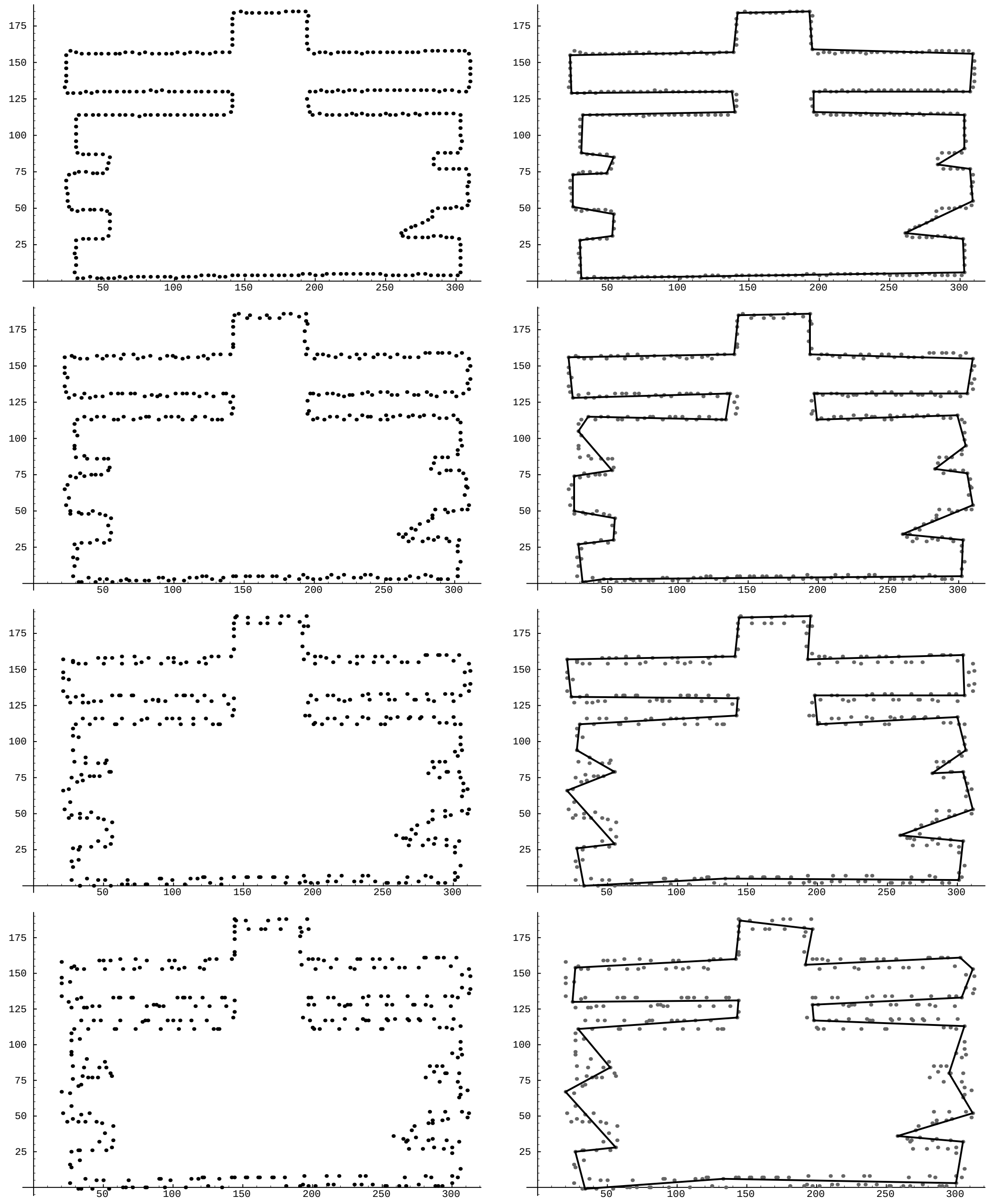

Figure 8: A Chinese character. On left from top to bottom: Increasing spatial noise (see text). On corresponding right: MDL Polygon laid over the point sequence.

to a shortest path problem in weighted graphs, a cleaner way to achieve this would be to use Floyd's all pair shortest path algorithm [10]. Also, various heuristics can be devised to obtain a feasible starting point. For example, a data point which, with its preceding and succeeding neighbors forms a nearly straight line, is a poor candidate for the starting point.

To summarize, we have presented with experimental results, a method which allows a fast and optimal—in the MDL sense–polygonal approximation to a given sequence of points in the Euclidean plane. Our experiments show the visually pleasing results that one gets using such an approach. We hope that such global and non-parametric methods would be of use both to the theoretician and to the practitioner of Machine/Computer Vision.

Acknowledgements

The authors gratefully acknowledge the assistance of various colleagues in the development of this paper: Byron Dom without whose comments, helpful suggestions, and careful reading the paper would not be what it is; Jorma Rissanen who helped us with our understanding of Minimum Description Length principles; Eduardo Rodriguez who supplied us with Figures 8 and inspired us; James G. Dunham who provided us the input data for Figure 7. We would also like to thank our numerous friends in Computer Science, IBM Almaden Research, who inspired us to carry out this work.

References

[1] T. Pavlidis. A review of algorithms for shape analysis. *Computer Graphics and Image Processing*, 7:243–258, 1978.

[2] T. Pavlidis. Algorithms for shape analysis of contours and waveforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(4):301–312, 1980.

[3] S. Marshall. Review of shape coding techniques. *Image and Vision Computing*, 7(4):281–294, 1986.

[4] J. G. Dunham. Optimum uniform piecewise linear approximation of planar curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):67–75, 1986.

[5] S. Itoh. An algorithm for the piecewise linear approximation of planar curves. In *Proceedings of the IEEE Symposium on Informantion Theory (ISIT) 1990*, page 62, 1990.

[6] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific, 1989.

[7] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 2, 1975.

[8] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 2(11):211–222, 1983.

[9] T. W. Anderson. *An Introduction to Multivariate Statistical Analysis.* Wiley, New York, second edition, 1984.

[10] G. Brassard and P. Bratley. *Algorithmics: Theory and Practice.* Prentice-Hall, 1988.