Linear Compression of Digital Ink via Point Selection

Vadim Mazalov and Stephen M. Watt
Department of Computer Science, University of Western Ontario, London, Canada

Abstract

We present a method to compress digital ink based on piecewise-linear approximation within a given error threshold. The objective is to achieve good compression ratio with very fast execution. The algorithm is especially effective on types of handwriting that have large portions with nearly linear parts, e.g. hand drawn geometric objects. We compare this method with an enhanced version of our earlier functional approximation algorithm, finding the new technique to give slightly worse compression while performing significantly faster. This suggests the presented method can be used in applications where speed of processing is of higher priority than the compression ratio.

Introduction

In this work we address the question of how to preserve the high precision of a curve, while decreasing the number of points representing it. The method can be viewed as a dynamic adjustment of the density of points, depending on the shape of a stroke. More points are removed from straighter regions than regions with high curvature. We have two subproblems that need to be solved:

1. decomposition of digital ink into inflection-free parts, and
2. compression of the individual pieces.

We measure the compression rate and time required to process the experimental datasets and compare with the performance of the enhanced functional compression. While losing in compression, the linear method is found to perform more than $100\times$ faster.

Functional Compression

The functional approximation technique is based on piecewise approximation of curves by truncated series in an orthogonal polynomial basis. In the past, we experimented with different orthogonal polynomial basis. We found Chebyshev polynomials to give the best results, and in this work, we improve their performance. The improvement is to be achieved by representing coefficients in a more compact form.

We consider the adaptive segmentation scheme. Coefficients are recorded as floating-point numbers with base 2. The significand and the exponent are two's complement binary integers, encoded in a and p bits respectively. The value of p is fixed, and the value of a is dynamically adjusted for each stroke. The following representation of each information channel of a trace i is proposed:

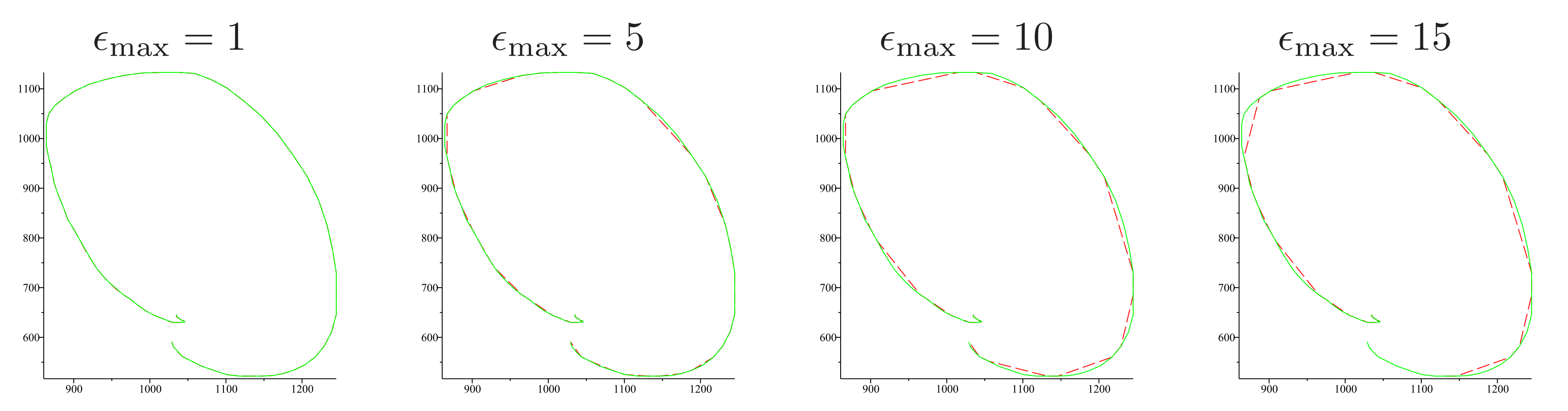
- Encode the 0 order coefficient in $2a+p$ bits, since this coefficient regulates the initial position of the trace and is typically larger than the rest of the coefficients. This number of bits is device-dependent.
- Find the coefficient $c_M = \max|c_i|, i = 1..d$ and encode it in $a+p$ bits.
- Encode coefficients $c_j, j = 1..d$, as two's complement binary integers $r_j = \left\lfloor \frac{|c_M|}{c_j} \right\rceil$ in b_r bits, where $\lfloor x \rceil$ represents rounding of x to the integer.

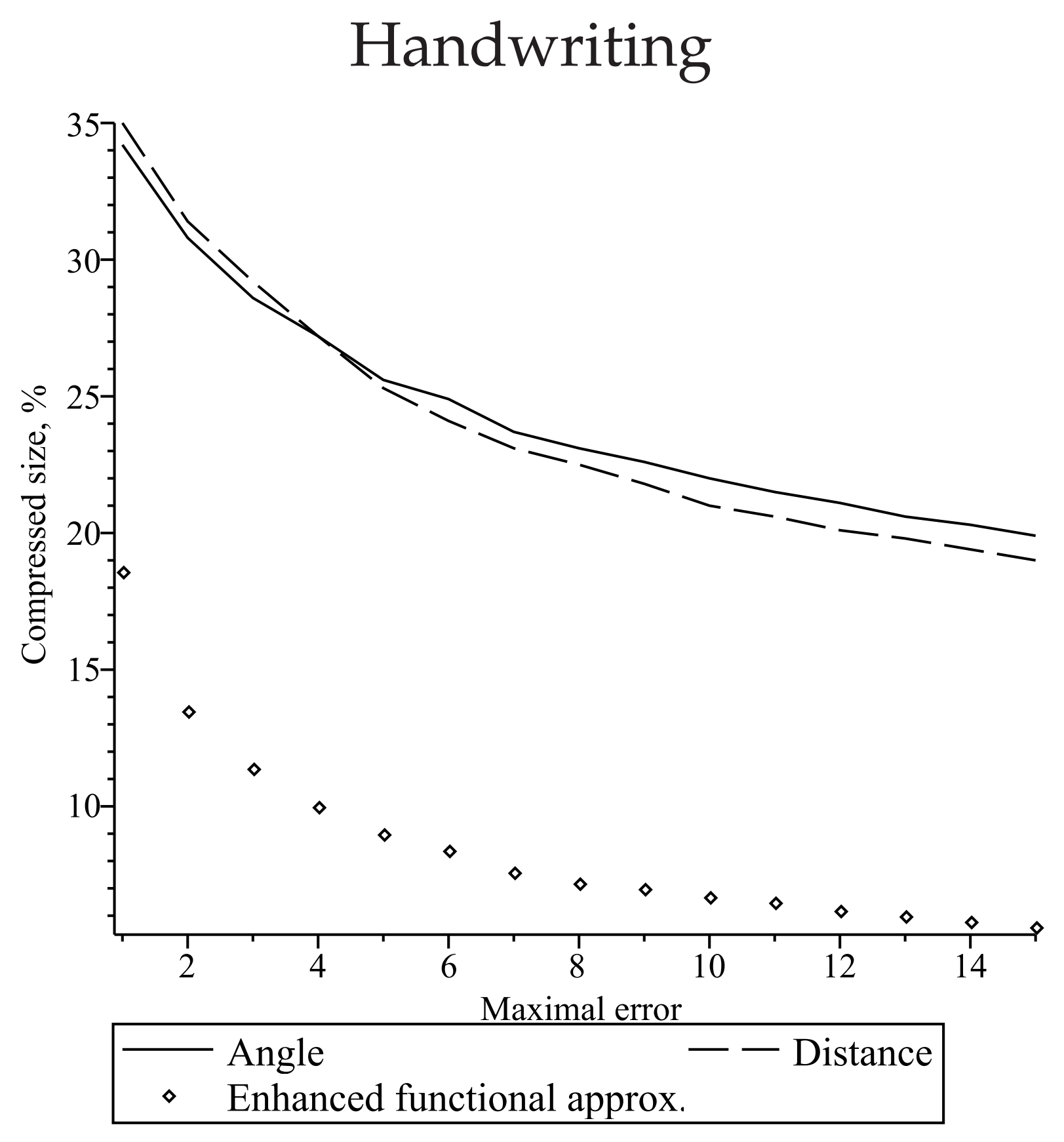
Thus, a trace i is recorded as

$$a_i d_i \lambda_1 c_{10} c_{1M} r_{11} \ldots r_{1d_i} \lambda_2 c_{20} c_{2M} r_{21} \ldots r_{2d_i} \ldots \lambda_D$$

where a_i is the number of bits for encoding the significand; d_i is the degree of approximation; λ_j is the initial value of parameterization of a piece j; c_{j0} is the 0-order coefficient; $c_{jM} = \max|c_{jk}|, k = 1..d$; $r_{jk} = \left\lfloor \frac{|c_{jM}|}{c_{jk}} \right\rceil$, c_{jk} is the k-th coefficient of the j-th piece. This differs from the previous method by having the coefficients c_j represented as scalings rounded to integers rather than as significand-exponent pairs.

Results





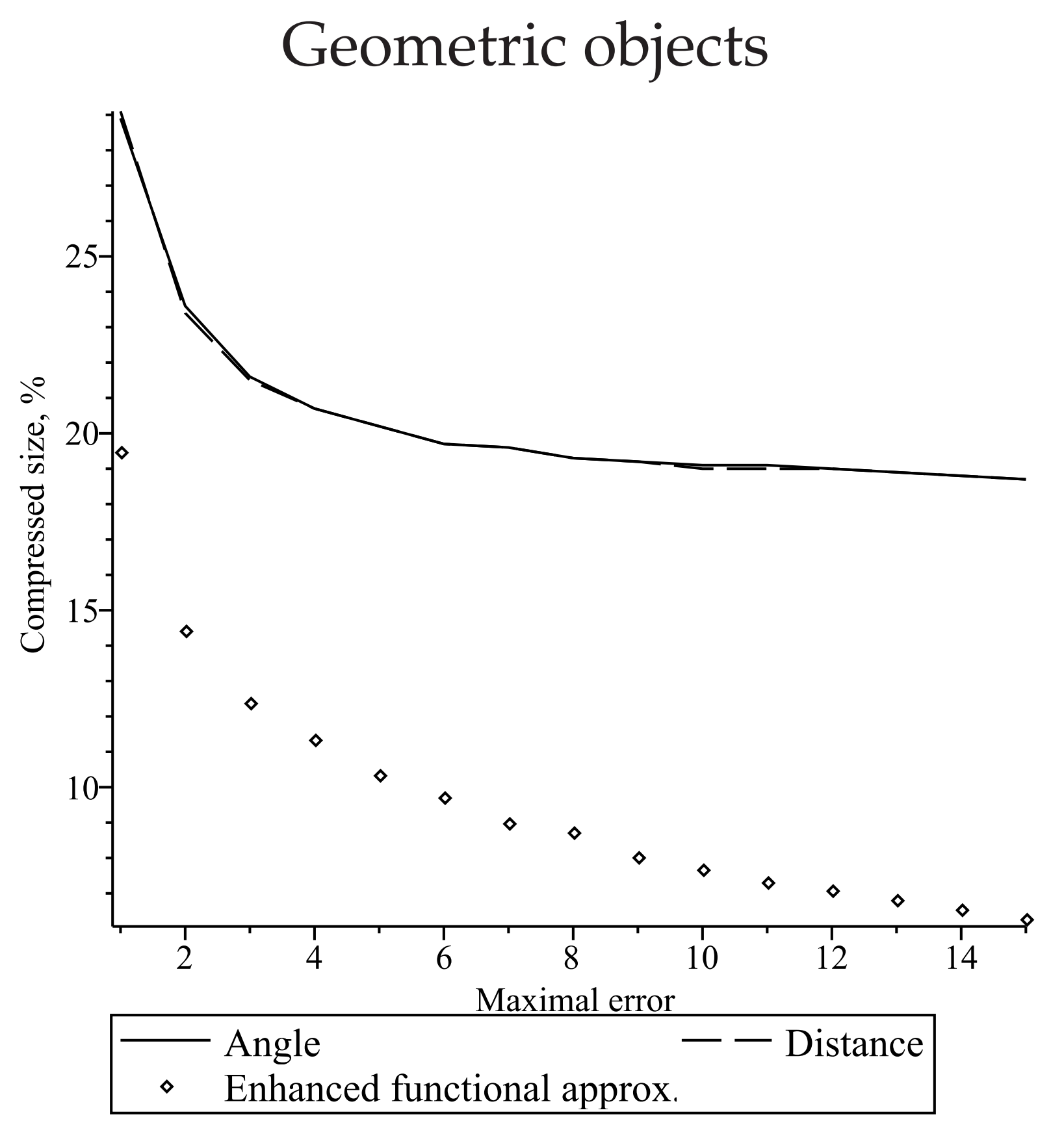


Table 1: Time (in seconds) for compression of the handwriting dataset

Method \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	25	20	21	21	17	17	17	19	18	15	16	15	15	20	16
F	879	1083	1287	1498	1700	1982	2188	2326	2479	2618	2727	2915	3019	3138	3327

Inflection-free parts

First, the curve should be decomposed into parts where the second derivative has constant sign, i.e the normal vector in the Frenet frame is pointing to the same side of the curve.

Algorithm 1 FormInflectionFreeSegments()

```
Input: Points – a stream of input points
Output: C – a list of inflection-free segments

  $C \leftarrow [\,]$ {list of inflection-free segments found}
  $S \leftarrow [\,]$ {current segment being collected}
  $i \leftarrow 0$ {index of current point without duplication}
  while Points.hasNext() do
    $P \leftarrow$ Points.getNext()
    if $i = 0$ or $P \neq P_{i-1}$ then
      $P_i \leftarrow P$
      if $|S| \geq 2$ then
        if $P_i = P_0$ then
          Append the list $S$ to the end of the list $C$
          $S \leftarrow [\,]$
        else
          $A_i \leftarrow \mathrm{Angle}(P_{i-2}, P_{i-1}, P_i) - \pi$
          $A_{\mathrm{Beg}} \leftarrow \mathrm{Angle}(P_i, P_0, P_1) - \pi$
          $A_{\mathrm{End}} \leftarrow \mathrm{Angle}(P_{i-1}, P_i, P_0) - \pi$
          if $A_i \times A_{i-1} < 0$
          or $A_i \times A_{\mathrm{End}} < 0$ or $A_{\mathrm{Beg}} \times A_{\mathrm{End}} < 0$ then
            Append the list $S$ to the end of the list $C$
            $S \leftarrow [\,]$
          end if
        end if
        Append $P_i$ to the end of the list $S$
        $i \leftarrow i + 1$
      end if
    end if
  end while
  If $S$ is non-empty, append it to the end of the list $C$
  return $C$
```

Compression of parts

If either the maximal error $||\cdot||_{\max}$ or the root mean square error $||\cdot||_{\mathrm{rms}}$ on an interval is greater than the respective thresholds $\epsilon_{\max}$ or ϵ_{rms}, the curve is split into two parts, see Algorithm 2.

Definition We write $\mathrm{pw}(L)$ for the piecewise linear curve defined by the list of points L. If two points a and b occur in a list L, with a preceding b, then we say that $[a, b]$ is an *interval* in L. We write $L|I$ for the sublist of L restricted to the interval I.

Algorithm 2 CompressCurve(S, R)

```
Input: $S$ – a list of points for an inflection-free segment
       $R$ – a partitioning rule (rule 1 or 2)
Output: $L$ – a list of points such that
       $||\mathrm{pw}(S) - \mathrm{pw}(L)||_{\max} < \epsilon_{\max}$ and
       $||\mathrm{pw}(S) - \mathrm{pw}(L)||_{\mathrm{rms}} < \epsilon_{\mathrm{rms}}$

  {$J$ is a stack of intervals to be refined.}
  $J \leftarrow$ [ Interval with first and last point of $S$ ]
  $L \leftarrow [\,]$
  while $J \neq [\,]$ do
    $j \leftarrow$ Pop an interval from $J$
    $a \leftarrow j$.first; $b \leftarrow j$.last
    if $||\mathrm{pw}(S|j) - \mathrm{pw}(j)||_{\max} > \epsilon_{\max}$
    or $||\mathrm{pw}(S|j) - \mathrm{pw}(j)||_{\mathrm{rms}} > \epsilon_{\mathrm{rms}}$ then
      {Split $j$ according to rule $R$ at some point $c$ in $S$}
      $j_1 \leftarrow [a, c]$
      $j_2 \leftarrow [c, b]$
      Push $j_2$ and then $j_1$ onto the stack $J$
    else
      Append $a$ and then $b$ to the end of list $L$
    end if
    Remove element $j$ from $J$
  end while
  return $L$
```

Conclusion

The decomposition algorithm processes each incoming point in constant time $O(1)$. The best case time complexity of compression of a piece is $O(n)$. If the splits are made unequally, always splitting n points as 1 and $n-1$, then the cost is $O(n^2)$.

Our experiments show the piecewise linear approximation method to perform about $100\times$ faster than the functional approximation algorithm, but it yields a less compact representation.