Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Pattern Recognition Letters 24 (2003) 2243–2254

Pattern Recognition Letters

www.elsevier.com/locate/patrec

Reduced-search dynamic programming for approximation of polygonal curves

Alexander Kolesnikov [a,b], Pasi Fränti [a,*]

[a] Department of Computer Science, University of Joensuu, P.O. Box 111, FIN-80101 Joensuu, Finland
[b] Institute of Automation and Electrometry, Pr.Ak.Koptyug, 1, Novosibirsk-90, 630090 Russia

Received 12 March 2002; received in revised form 21 November 2002

Abstract

Approximation of polygonal curves with minimum error (*min-ε problem*) can be solved by dynamic programming, or by graph-theoretical approach. These methods provide optimal solution but they are slow for a large number of vertices. Faster methods exist but they lack the optimality. We try to bridge the gap between the slow but optimal, and the fast but sub-optimal algorithms by giving a new near-optimal approximation algorithm based on reduced-search dynamic programming. The algorithm can be iterated as many times as further improvement is achieved in the optimization. It is simple, fast, and it has a low space complexity.

Keywords: Polygonal approximation; Dynamic programming; Data reduction; Vectorization

1. Introduction

Approximation of polygonal curve is an important task in computer vision, computer graphics, digital cartography, and data compression. The problem is defined as follows: Given an open N-vertex polygonal curve P, approximate it by another polygonal curve Q with a given number of segments M so that the approximation error is minimized. This problem formulation is known as the *min-ε problem*.

There are two well-known approaches for solving the problem: *graph-theoretical* and *dynamic programming*. Graph-theoretical algorithms generate first a weighted *directed acyclic graph* (DAG) on the vertices of P, and then find the shortest path in the graph (Imai and Iri, 1986, 1988; Melkman and O'Rourke, 1988; Chan and Chin, 1996; Zhu and Seneviratne, 1997; Chen and Daescu, 1998; Katsaggelos et al., 1998; Schuster and Katsaggelos, 1998). This can be solved in $O(N^2 \log N)$ time (Chan and Chin, 1996) in $O(N)$ space (Chen and Daescu, 1998). In real-time applications, however, the method is useful only for small values of N because of the high time complexity.

Dynamic programming generates the solution for the problem recursively using the results of the smaller problem instances (Perez and Vidal, 1994).

* Corresponding author. Tel.: +358-13-251-7931; fax: +358-13-251-7955.
E-mail addresses: koles@cs.joensuu.fi (A. Kolesnikov), franti@cs.joensuu.fi (P. Fränti).

0167-8655/03/$ - see front matter
doi:10.1016/S0167-8655(03)00051-5

Straightforward implementation of this requires $O(MN^2)$ time and $O(MN)$ space. Salotti has improved this approach by a method that has time complexity close to $O(N^2)$ (Salotti, 2000–2002). These approaches, however, are useful only for a relatively small values of N because of the time and memory requirements.

There also exist a number of faster but sub-optimal methods (Douglas and Peucker, 1973; Ray and Ray, 1994; Pikaz and Dinstein, 1995; Rosin, 1997; Yin, 1998; Huang and Sun, 1999; Zhang and Guo, 2001) with time complexities ranging from $O(N)$ to $O(N^2)$. The quality of the sub-optimal methods, however, usually remains less than 80% in comparison to that of the optimal solution (Rosin, 1997). In the applications such as vector map data reduction in geographical information systems (GIS), the number of vertices can be very high, e.g., from $N = 10^3$ to 10^4, and $M = 10^2$ to 10^3, respectively. For example, high-quality vectorization of digitized curve requires pixel-level accuracy in the processing (Wenyin and Dori, 1999; Fränti et al., 1999). Low time and space complexities are therefore important in applications using large-scale data.

In this paper, we try to bridge the gap between the slow but optimal, and the fast but sub-optimal algorithms by proposing a new near-optimal algorithm based on the dynamic programming with iterative reduced search in the state space. At first step, an initial solution is generated using any fast sub-optimal algorithm. The solution defines a reference path in the state space. *A bounding corridor* is then constructed along this path, and the minimum cost path is searched within the corridor using dynamic programming. The method explores only a small but relevant part of the state space. The algorithm can be iterated using the output solution as a new reference path in the next iteration. The proposed algorithm is simple, fast, and it can be implemented in $O(N)$ space. The time and quality trade-off can be controlled by setting up an appropriate corridor width.

2. Optimization problem formulation

Let us define an *open N-vertex polygonal curve P* in two-dimensional space as the ordered set of vertices $P = \{p_1, p_2, \ldots, p_N\} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$. The problem can be stated as follows: approximate the polygonal curve P by another polygonal curve Q with a given number of line segments M so that total approximation error E is minimized for a given error measure. The output polygonal curve Q consists of $(M + 1)$ vertices: $Q = \{q_1, \ldots, q_{M+1}\}$, where the set of vertices q_m is a subset of P. End points of Q are end points of $P : q_1 = p_1$, and $q_{M+1} = p_N$. The approximation line segment (q_m, q_{m+1}) for curve segment $\{p_i, \ldots, p_j\}$ is defined by the end points p_i and p_j: $q_m = p_i$ and $q_{m+1} = p_j$.

The error of the approximation of curve segment $\{p_i, \ldots, p_j\}$ of P with the corresponding line segment (q_m, q_{m+1}) of Q is defined here as the sum of squared Euclidean distances from each vertex of $\{p_i, \ldots, p_j\}$ to the corresponding line segment (q_m, q_{m+1}):

$$e^2(p_i, p_j) = \sum_{k=i+1}^{j-1} (y_k - a_{ij}x_k - b_{ij})^2/(1 + a_{ij}^2), \qquad (1)$$

where the coefficients a_{ij} and b_{ij} are defined from the linear equation $y = a_{ij}x + b_{ij}$ of the line segment (p_i, p_j). The error $e^2(p_i, p_j)$ of L_2 metrics can be calculated in $O(1)$ time with five arrays of the cumulative sums for x, y, x^2, y^2, xy as in the incremental algorithm by Perez and Vidal (1994). The time required for the calculation of the approximation error $e^2(p_i, p_j)$ is independent on the number of vertices in the curve segment $\{p_i, \ldots, p_j\}$.

The total approximation error of the input polygonal curve P by the output polygonal curve Q is the sum of the errors of approximating each segment $\{p_i, \ldots, p_j\}$ of P by the corresponding line segment (q_m, q_{m+1}) of Q:

$$E = \sum_{m=1}^{M} e^2(q_m, q_{m+1}). \qquad (2)$$

The optimal approximation of P is then the set of vertices $\{q_2, \ldots, q_M\}$ that minimizes the cost function E:

$$E = \min_{\{q_m\}} \sum_{m=1}^{M} e^2(q_m, q_{m+1}) \qquad (3)$$

To solve the optimization task we first recall the optimal dynamic programming algorithm of Perez and Vidal (1994) with minor modifications in Section 3. We then introduce the new near-optimal iterative reduced-search algorithm in Section 4.

3. Dynamic programming approach

Let us first define a discrete two-dimensional *state space* $\Omega = \{(n,m) : n = 1,\ldots,N; m = 0,\ldots,M\}$. Every point (n,m) in the state space Ω represents the sub-problem of approximating of an n-vertex polygonal curve $\{p_1,p_2,\ldots,p_n\}$ by m line segments. The complete problem is represented by the goal state (N,M).

An output polygonal curve Q can be represented as a *path* $G = \{g(0),g(1),\ldots,g(M)\}$ in the state space Ω from the start state $(1,0)$ to the goal state (N,M). In the state space, we can also define a function $D(n,m)$ of the state (n,m) as the cost function value of the optimal approximation for the n-vertex curve $\{p_1,p_2,\ldots,p_n\}$ by m line segments.

3.1. Full-search algorithm

For solving the *min-e problem* under question we have to find the optimal path to the goal state (N,M). We can reduce the state space Ω by a left bound L, and by a right bound R as follows (see Fig. 1 left part):

$$L(m) = \begin{cases} m+1; & m = 0,1,\ldots,M-1; \\ N; & m = M; \end{cases}$$

$$R(m) = \begin{cases} 1; & m = 0; \\ N-M+m; & m = 1,2,\ldots,M. \end{cases}$$

We also introduce a bottom $B(n)$ and top $T(n)$ bounds to represent the state space Ω as a bounded space:

$$B(n) = \begin{cases} 0; & n = 1, \\ 1; & n = 2,\ldots,N-M \\ n-(N-M)+1; & n = N-M+1,\ldots,N; \end{cases}$$

$$T(n) = \begin{cases} n-1; & n = 1,\ldots,M-1; \\ M-1; & n = M,\ldots,N-1; \\ M; & n = N. \end{cases}$$

The optimization problem can be solved by the dynamic programming algorithm as proposed by Perez and Vidal (1994) with the following recursive expressions:

$$D(n,m) = \min_{L(m-1)\leqslant j<n} \{D(j,m-1) + e^2(p_j,p_n)\};$$
$$A(n,m) = \arg\min_{L(m-1)\leqslant j<n} \{D(j,m-1) + e^2(p_j,p_n)\}. \quad (4)$$

Here $A(n,m)$ is the *parent state* that provides the minimum value for the cost function $D(n,m)$ at the state (n,m).

It was mentioned by Perez and Vidal (1994) that the calculation of these formulas in the state space Ω can be performed in two alternative ways: column-by-column, or row-by-row. In their algorithm, Perez and Vidal used latter one: row-by-row order. In this case, the approximation errors must be calculated "on-fly", because otherwise we had to use a two-dimensional array of size $N \times N$ to store the calculated values for further use.

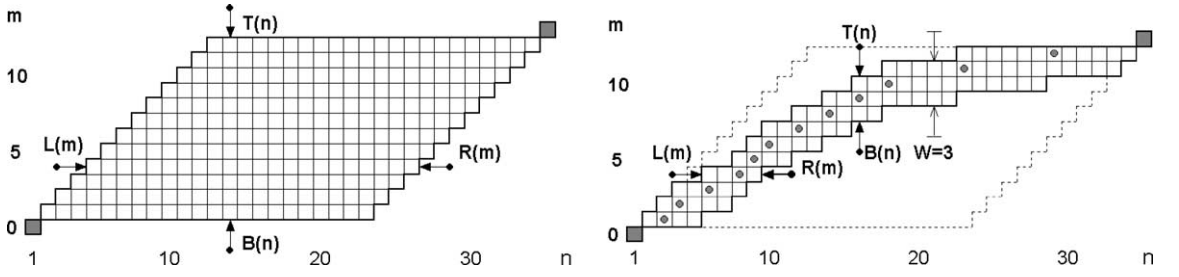

Fig. 1. *Left*: illustration of the state space Ω for a sample problem size of $N = 35$, $M = 13$. The start and goal states are marked with the gray squares. *Right*: illustration of the bounding corridor of width $W = 3$ in the state space Ω. The reference path G is marked by the gray circles; the full state space Ω is marked by the dashed line.

```
// Search in the bounded state space
D(1,0) ←0
FOR  n = 2  TO  N  DO
   // Calculation of the approximation errors
   FOR j = L(B(n) – 1) TO  n–1  DO
       v(n–j) ← e²(pj,pn)
   END

  // Search of the minimum
   FOR m = B(n)  TO  T(n)  DO
      dmin ← ∞
      FOR  j= L(m–1) TO  n–1  DO
          d ← D(j, m–1–B(n)) + v(n–j)
          IF(d < dmin)
             dmin ← d;
             jmin ← j
          ENDIF
       END
       D(n, m–B(n)) ← dmin
       A(n, m–B(n)) ← jmin
   END
END
E ← D(N,M–B(N))
```

Fig. 2. General scheme of the minimum search in the bounded state space.

We take the other column-by-column order. In this scheme, the outmost loop is performed on the vertex number n instead of the segment number m (see Fig. 2). We can see that the approximation error $e^2(p_j, p_n)$ does not depend on the number of segments m. The errors can therefore be calculated before the m loop and then stored in a *one-dimensional* array $v(n)$ of size N. We can calculate every value of $e^2(p_j, p_n)$ only once and avoid further recalculations of the same values. We consider next how this pre-calculation scheme affects the processing time and the complexity of the algorithm.

3.2. Time and space complexity

Let us consider the time complexity of the modified full search dynamic programming algorithm as represented in Fig. 2. The computation time of the algorithm consists of two parts: calculation of the errors, and the search of the minimum.

The time complexity of the error calculation is $O(N^2)$: for every vertex $p_n (n = 1, \ldots, N)$ we have to calculate n values of the approximation errors. The time complexity of the minimum search procedure is $O(M(N-M)^2)$: we have to calculate the cost function value for $M(N-M)$ locations, and each of them requires at most $(N-M)$ operations.

The total time complexity of the algorithm for error metrics L_2 is dominated by the second step, and it is therefore $O(M(N-M)^2)$. In the special case, when the number of line segments is close to the number of the input vertices ($M = N - \Delta n$; $\Delta n = 1, 2, \ldots$), the time complexity of the algorithm is $O(N)$. In practical situations, however, when the number of segment is proportional to the number of vertices ($M \sim N/c$), the time complexity of the algorithm is $O(N^3)$. For small values of $M (M \ll N)$ the processing time is proportional to MN^2. Thus, the proposed scheme (see Fig. 2) does not reduce the time complexity in comparison to that of the algorithm by Perez and Vidal, but it can reduce the processing time in practice.

The space complexity of the represented algorithm is defined mostly by the memory requirements for storing the cost function values $D(n,m)$, and the parent function $A(n,m)$. These equals to $O(MN)$.

3.3. Bounding the search

The main reason of the high time complexity is the search redundancy of the dynamic programming approach. The key issue is that with dynamic programming method the *global* minimum of the cost function is searched in a *sequential* manner from the starting state to the goal state. This is because we do not know in advance which sub-problem solutions exactly will be used to construct the final optimal solution, so we have to find all of them in the state space to get the final solution.

Salotti (2000) has proposed a rough sub-optimal approximation to partially solve this problem by obtaining an upper limit for the total error of approximation. In his algorithm, the state space is processed at each stage only until the total error exceeds the given upper limit. In this way, part of the state space can be excluded without any computation. Salotti proposed two methods to estimate the lower limits for the error. These ideas allow to reduce the time complexity to $O(N^2)$ according to Salotti (2000, 2001, 2002).

Even though the optimal algorithm by Salotti is faster than the original one by Vidal and Perez the, time complexity can still be too high for large values of N and M to be used if the time is limited.

4. Iterative reduced-search dynamic programming

Based on the dynamic programming, we next introduce a new iterative search method that explores only a small but relevant part of the state space. The key idea of the method is to have a rough estimation for the *path* of the optimal solution in the state space. We can then reduce the search to a limited state space along the *reference path*. The search can be iterated starting from the new reference solutions as many times as desired to achieve further improvement of the solution.

4.1. Reduced-search algorithm

The proposed algorithm consists of the following three steps:

Step 1: Find a reference solution by any fast heuristic algorithm.
Step 2: Construct a bounding corridor in the state space.
Step 3: Apply dynamic programming within the bounding corridor.

Let us next consider the steps in detail.

Step 1: Any fast sub-optimal algorithm can be used for finding the reference solution. We use the *Douglas–Peucker* algorithm (Douglas and Peucker, 1973) of time complexity $O(MN)$. The obtained solution defines a reference path $G = \{g(0),\ldots,g(M)\}$ in the state space Ω.

Step 2: A bounding corridor of width W is constructed along the reference path G in the space Ω (see Fig. 1). The left $L(m)$, right $R(m)$, bottom $B(n)$, and top $T(n)$ bounds of the corridor are defined as follows:

$$L(m) = \begin{cases} m+1; & m = 0,\ldots,c_1, \\ \max\{m+1,\, g(m-c_1)\}; & m = c_1+1,\ldots,M, \end{cases}$$

$$R(m) = \begin{cases} \min\{N, g(m+c_2)-1\}; & m = 0,\ldots,M-c_2, \\ N; & m = M-c_2+1,\ldots,M, \end{cases}$$

$$B(n) = \begin{cases} 0; & n = 0, \\ m; & n = R(m-1)+1,\ldots,R(m), \end{cases}$$

$$T(n) = \begin{cases} \min\{M, m+W-1\}; & n = L(m),\ldots,L(m+1)-1; \\ M; & n = N. \end{cases}$$

Here $c_1 = \lfloor W/2 \rfloor$, $c_2 = W - c_1$.

Step 3: Dynamic programming is performed inside the bounding corridor for solving the minimum cost function $D(n,m)$ using the recursive expression of Eq. (4). This part of the algorithm

(referred here as the *optimization step*) is outlined in Fig. 2.

4.2. Iterations

The algorithm can be iterated using the output solution as a new reference path in the next iteration. The number of iterations can be given in advance, or adaptively varied depending on the development of the approximation error.

4.3. Complexity of the algorithm

We next estimate the time complexity of the algorithm under the following assumptions: $N \gg 1$ and $W \leqslant M$. As in the case of the full search algorithm, the processing time of the search in the corridor consists of two parts: the calculation of the approximation errors, and the calculation of the cost function $D(n,m)$.

Overall, we have to calculate no more than WN^2/M values of the approximation error (WN/M values for N vertices). Then for WM states in the bounding corridor we must perform WN/M operations (on average) to find the minimum value of the cost function. In total, we have W^2N^2/M operations of the complexity of O(1) each. Thus, as in the case of the full search, the time complexity of the proposed algorithm is defined by the complexity of the minimum search procedure, and the total number of operations is proportional to N^2W^2/M per iteration. The expected speed-up of the reduced-search approach is proportional to $(W/M)^2$ in comparison to the time complexity of the full search.

The time complexity of the proposed algorithm is summarized in Table 1 and compared with that of the full search. The time complexity has an inverse linear dependency to the number of segments (M), and therefore, the optimization algorithm is faster for larger values of M. The lowest value is $O(N)$ in the case when M is proportional to N, and the highest value is $O(N^2)$ in the case when M is a small constant. In both cases, the proposed algorithm is an order of magnitude faster than the full search dynamic programming. For example, when $W = 10$ and $M = 100$, the speed-up of the reduced search algorithm for a polygon of $N = 5000$ vertices is about 100. To get overall processing time of the proposed algorithm, we have to add the time required for obtaining the initial (reference) solution.

In the implementation, we use two arrays of size $W \times N$: one array for storing the values of the costs values $D(n,m)$, and another array for storing the parent states $A(n,m)$. Thus, the total memory requirement of the algorithm is proportional to $2WN$, which is $O(WN)$. Thus, the reduction of the space complexity is about W/M in comparison to that of the full search.

5. Experimental results and discussion

In order to evaluate the quality of the sub-optimal algorithms, Rosin (1997) introduced a measure known as *fidelity* (F). It measures how good a given sub-optimal approximation is in respect to the optimal approximation in terms of the approximation error:

$$F = \frac{E_{\min}}{E} \times 100$$

Here $E_{\min}$ is the approximation error of the optimal solution, and E is the approximation error of the given sub-optimal solution.

We next study the performance of the algorithm by measuring the fidelity and run time. The minimum approximation error $E_{\min}$ is obtained by using the algorithm of Perez and Vidal. We use the

Table 1
Time complexities of the full search and reduced search dynamic programming

	Time complexity	Lower limit ($M \leqslant N/c$)	Upper limit ($M \geqslant c$)
Full search	$O(N^2M)$	$\Omega(N^3)$	$O(N^2)$
Reduced search	$O(N^2W^2/M)$	$\Omega(N)$	$O(N^2)$

Here c is defined as a small constant.

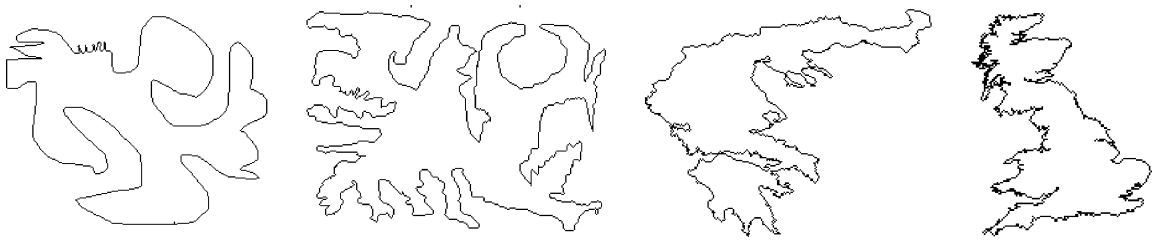

Fig. 3. The set of test shapes: #1: digitized curve from (Salotti, 2001), $N = 3222$. #2: digitized curve from (Salotti, 2001), $N = 5004$. #3: vector contour of Greece, $N = 5542$. #4: vector contour of the Great Britain, $N = 10,910$.

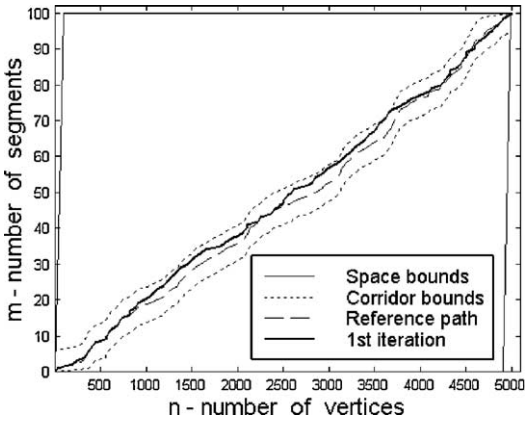

Fig. 4. Illustration of the bounding corridor ($W = 10$) in the state space for approximation of the test shape #2 with 100 segments ($M = 100$). The reference path is obtained by the Douglas–Peucker algorithm.

test shapes shown in Fig. 3. The test shapes #1 and #2 are digitized curves from (Salotti, 2001), the other two shapes are vector contours from the ESRI Map Set.

The processing time and fidelity of the solution depend on the following parameters: the corridor width (W), the number of iterations (k), and the fidelity of the reference solution (F_0). We can regulate the trade-off between the quality and run time by changing the values of these parameters. By using a narrow corridor (small value of W) the algorithm is fast. On the other hand, more accurate approximation is expected when using a wider corridor but at the cost of a slower algorithm. The quality can also be improved by using several iterations but at the cost of a higher run time. It is also expected that better results can be obtained faster when using a high quality reference solution as input.

5.1. The corridor width and the number of iterations

We first measure the fidelity and run time of the proposed algorithm for the shape #2 (see Fig. 3) using the corridor widths from $W = 2$ to 12. The results are shown in Tables 2 and 3, and illustrated in Figs. 4 and 5. The Douglas–Peucker algorithm is used to generate the reference solution. The run times have been obtained using Pentium II, 266 MHz processor.

The fidelity of the reference solution (Douglas–Peucker) is about $F_0 = 50$–60%. The fidelity of the proposed method is about $F_1 = 95$–100% after the 1st iteration (Table 2), except the narrowest corridor width $W = 2$. After the 2nd iteration (Table 3), the fidelity reaches values of about $F_2 = 95$–100% for $W \geqslant 4$, and the algorithm usually converges after 3–6 iterations. The only exception is the narrowest corridor ($W = 2$), for which the cost function converges to a local minimum and reaches fidelity of about 90–99%. It is also noted that the measured run times correlates very well to the time complexity function $T = N^2W^2/M$.

5.2. Quality of the reference approximation

In order to study the sensitivity of the algorithm to the quality of the reference solution, we generated also a *random* solution by selecting $(M - 1)$ random points from the test shape #2. The most relevant results are summarized in Fig. 6 using both the Douglas–Peucker and the random solution as the reference solution. The fidelity of the

Table 2
Computation time (T_1) in seconds and fidelity values (F_1) after the 1st iteration of the proposed algorithm for the test shape #2

Shape #2	$M = 50$		$M = 100$		$M = 300$		$M = 500$		$M = 700$	
	T	F	T	F	T	F	T	F	T	F
$W = 2$	0.6	97.0	0.4	83.3	0.1	89.8	0.1	90.5	0.1	88.3
$W = 4$	1.8	99.3	1.2	95.2	0.4	97.7	0.2	97.9	0.2	96.8
$W = 6$	3.7	100	2.4	97.6	0.8	98.9	0.5	99.7	0.3	98.9
$W = 8$	6.4	100	3.8	98.8	1.2	99.7	0.8	99.9	0.6	99.6
$W = 10$	8.9	100	5.7	99.3	1.9	99.9	1.2	99.9	0.8	99.7
$W = 12$	13.4	100	7.9	99.8	2.6	100	1.6	100	1.2	99.9

The computation time T_1 does not include the time cost of the preliminary approximation.

Table 3
Computation time (T_2) in seconds and fidelity values (F_2) after the 2nd iteration of the proposed algorithm for the test shape #2

Shape #2	$M = 50$		$M = 100$		$M = 300$		$M = 500$		$M = 700$	
	T	F	T	F	T	F	T	F	T	F
$W = 2$	1.2	99.3	0.7	91.1	0.2	94.8	0.2	93.9	0.1	93.1
$W = 4$	3.8	100	2.3	98.5	0.7	99.2	0.5	99.6	0.3	99.0
$W = 6$	7.5	100	4.7	98.8	1.5	100	0.9	99.9	0.7	99.8
$W = 8$	12.7	100	7.9	99.8	2.5	100	1.6	100	1.1	100
$W = 10$	18.1	100	11.6	100	3.8	100	2.3	100	1.6	100
$W = 12$	25.8	100	15.7	100	5.3	100	3.3	100	2.3	100

The computation time T_2 does not include the time cost of the preliminary approximation.

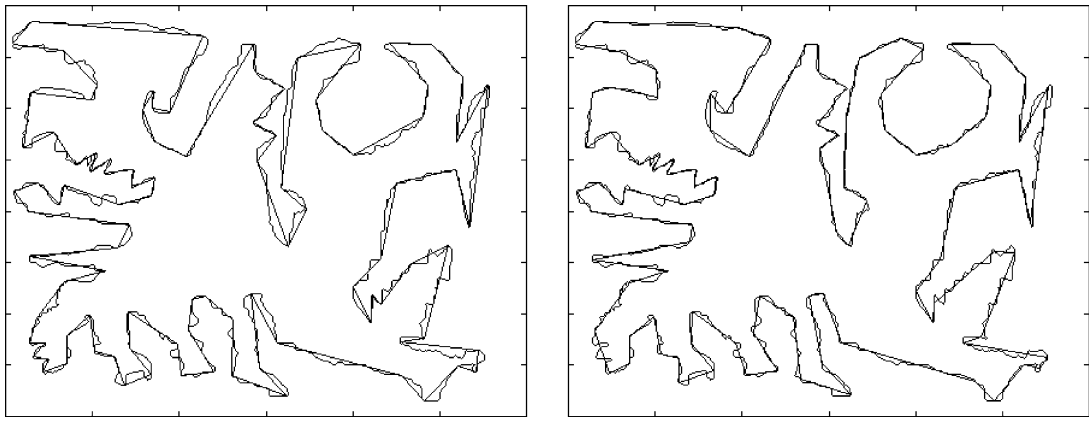

Fig. 5. *Left*: result of the approximation for test shape #2 with $M = 100$ segments by the Douglas–Peucker algorithm. The fidelity of the reference solution $F_0 = 41.7\%$, processing time $T_0 = 0.5$ s. *Right*: result of the approximation after the 1st iteration using corridor width of $W = 10$. The fidelity is $F_1 = 99.3\%$, and the processing time $T_1 = 6.2$ s (including the time spent for the initial approximation).

random approximation is very low for obvious reasons ($F_0 = 1$–5%), but the proposed algorithm achieves high fidelity values after a few iterations of optimization. Thus, the proposed algorithm reaches near-optimal (or optimal) result even if the quality of the reference solution is poor. The algorithm just needs few more iterations when starting from a random solution.

The complexity of the Douglas–Peucker algorithm is $O(MN)$, and the complexity of the pro-

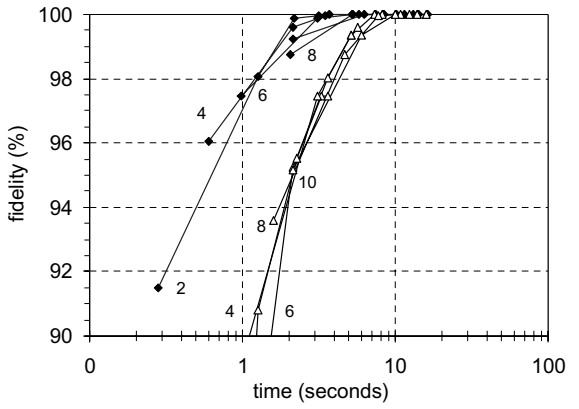

Fig. 6. Time–fidelity comparison of the proposed algorithm using the test shape #2 with $M = 200$ segments. The reference solution is generated by the Douglas–Peucker algorithm ($F_0 = 50.6\%$) (♦), and by random choice ($F_0 = 1.8\%$) (▵). The results in each line are after the 1st, 2nd, 3rd, 4th and 5th iterations, and the lines are for corridor widths from $W = 2$ to 10.

posed algorithm is $O(N^2W^2/M)$. The total processing time is the sum of the processing times of generating the reference approximation and of the optimization. In the case of a relatively large number of segments, the processing time of the Douglas–Peucker approximation is greater than that of a single iteration of the proposed optimization algorithm (see Table 4). In this case, a rough approximation by vertices decimation from N to $(M + 1)$ could be used to obtain a reference solution instead of the Douglas–Peucker algorithm; or another heuristic algorithm of the same complexity could be used. For the sake of simplicity, we restrict our studies here with the Douglas–Peucker algorithm for all values of M.

5.3. The choice of the parameters

On the basis of the previous experiments, we can see that the outcome of the algorithm depends on the choice of the parameters. The corridor width can be set to anything from 0 and M (reasonable values being from $W = 2$ to 10), and the algorithm usually iterates 3 to 6 iterations before converging. The corresponding fidelity values of the solutions vary from $F = F_0$ (no optimization) to $F_k = 100\%$ ($W = M$). We next recommend three feasible parameter set-ups for achieving the following goals:

- Sub-optimal algorithm (SA): minimize the time; fidelity is less important.
- Near-optimal algorithm (NA): maximize fidelity in a reasonable time.
- Practically optimal algorithm (POA): aim at optimal result ($F = 100\%$) with high confidence.

5.3.1. Sub-optimal algorithm

The algorithm performs only one step of the optimization using a narrow corridor. The recommended parameters for SA are: $k = 1$ and $W_1 = 2$.

5.3.2. Near-optimal algorithm

The algorithm performs a fixed number of iterations (2–3) using a wider corridor ($W = 6$–8) aiming at a high fidelity (of about $F = 99$–100%) in a reasonable time. The corridor must be wide enough in order to avoid local minimum but not too wide because the processing time is a quadratic function (W^2) of the corridor width (W). To regulate the processing time, the first iteration is

Table 4
Comparison of the processing time (T) in seconds and fidelity (F) of the three variants of the proposed algorithm (SA, NA, POA), and three existing methods (D–P, Salotti, full search) for the test shape #1 ($N = 3222$)

Shape #1	$M = 40$		$M = 60$		$M = 80$		$M = 100$		$M = 150$	
	T	F	T	F	T	F	T	F	T	F
D–P	0.1	43.3	0.1	31.9	0.2	49.0	0.2	48.4	0.4	56.2
New: SA	0.7	81.2	0.5	74.4	0.5	94.0	0.5	84.7	0.5	89.6
New: NA	3.6	100	2.3	98.5	1.8	100	1.4	99.6	0.9	100
New: POA	6.0	100	6.1	100	3.8	100	3.1	100	2.3	100
Salotti	60	100	59	100	58	100	58	100	76	100
Full search	60	100	94	100	310	100	470	100	535	100

The processing times of the SA, NA and POA include the time required for generating the reference approximation by D–P.

performed with a narrower corridor than the second one. The recommended parameters for the NA are: $k = 2$, for the test shape #1 we have $W_1 = 4$ in the 1st iteration and $W_2 = 6$ in the 2nd iteration; $W_1 = 6$ and $W_2 = 8$ for the shapes #2, #3, and #4.

5.3.3. Practically optimal algorithm

In principle, the proposed algorithm achieves optimal result when the corridor width is set to $W = M$, as it then performs full search in the entire state space Ω. In practice, the optimal result can be reached by using a much smaller corridor. A better strategy is therefore to change the parameters during the iterations as follows.

The algorithm first performs iterations using an initial corridor width W_1 until no change is achieved in the approximation error E_1. The algorithm is then iterated using a wider corridor $W_2 = W_1 + 2$ to get a new result with error E_2. If the approximation error keeps reducing ($E_2 < E_1$), the procedure is then repeated for wider corridor widths $W_{t+1} = W_{t+2}$ as long as further improvement is obtained. In other words, we continue to enlarge the corridor width and repeat the iterations until the optimized solution does "feel" the borders of the bounding corridor.

It is expected that the POA achieves the optimal result in almost all cases. In fact, we have not found any examples yet in which the algorithm does not converge to the optimal result. Nevertheless, the open question remains: "Does the strategy guarantee to find the global minimum of approximation error for *any* polygonal curve?". This is interesting question from the theoretical point of view. On the other hand, the suggested fast near-optimal algorithm using iterative reduced-search is able to satisfy the time demands in real-time systems in GIS, vectorization task, and pattern recognition applications.

On the basis of the previous experiments, we recommend the following parameters: $W_1 = 4$ for the curve #1, $W_1 = 6$ for shapes #2, #3, and #4; the number of iterations depends on characteristics of the curve.

5.4. Comparison to other algorithms

The results of the three variants (SA, NA, POA) for the test shapes #1 to #4 are summarized in Tables 4–7, and compared with the sub-optimal algorithm by Douglas–Peucker (D–P), the optimal algorithm by Salotti (2001), and the full search dynamic programming approach (Perez and Vidal, 1994). The optimal algorithm was implemented with the computational scheme of re-using the precalculated errors as explained in Section 3 (see Fig. 2). It allows us to reduce the processing time for the original full search dynamic programming in comparison to the straightforward implementation. Note that the results of the Salotti's method are given only for the shapes #1 and #2 because the software (given by the author) does not support shape files with float point representation as is the case with shapes #3 and #4. In most cases, the proposed algorithm achieves optimal or near-optimal results quite fast.

Table 5
Comparison of the processing time (T) in seconds and fidelity (F) of the three variants of the proposed algorithm (SA, NA, POA), and three existing methods (D–P, Salotti, full search) for the test shape #2 ($N = 5004$)

Shape #2	$M = 50$		$M = 100$		$M = 300$		$M = 500$		$M = 700$	
	T	F	T	F	T	F	T	F	T	F
D–P	0.3	45.5	0.5	41.7	1.0	50.2	2.4	58.0	3.4	62.0
New: SA	0.9	97.0	0.9	83.3	1.1	89.8	2.5	90.5	3.5	88.3
New: NA	12.3	100	7.0	99.8	3.0	100	3.6	100	4.3	100
New: POA	11.3	100	13.5	100	4.4	100	5.1	100	5.4	100
Salotti	150	100	190	100	250	100	390	100	700	100
Full search	216	100	506	100	1090	100	1720	100	2200	100

The processing times of the SA, NA and POA include the time required for generating the reference approximation by D–P.

Table 6
Comparison of the processing time (T) in seconds and fidelity (F) of the three variants of the proposed algorithm (SA, NA, POA), and two existing methods (D–P, full search) for the test shape #3 ($N = 5541$)

Shape #3	$M = 50$		$M = 100$		$M = 300$		$M = 500$		$M = 700$	
	T	F	T	F	T	F	T	F	T	F
D–P	0.2	52.4	0.4	51.7	1.2	52.3	2.1	54.3	3.0	62.4
New: SA	1.8	96.1	1.2	89.5	1.4	89.5	2.3	89.7	3.1	92.0
New: NA	14.9	100	8.5	100	3.3	99.7	3.5	99.0	3.9	99.1
New: POA	13.0	100	7.9	100	4.9	100	5.5	100	7.6	100
Full search	278	100	670	100	1830	100	3590	100	5425	100

The processing times of the SA, NA and POA include the time required for generating the reference approximation by D–P.

Table 7
Comparison of the processing time (T) in seconds and fidelity (F) of the three variants of the proposed algorithm (SA, NA, POA), and two existing methods (D–P, full search) for the test shape #4 ($N = 10,910$)

Shape #4	$M = 50$		$M = 100$		$M = 300$		$M = 500$		$M = 700$	
	T	F	T	F	T	F	T	F	T	F
D–P	0.4	46.3	0.87	36.9	2.5	38.5	4.2	40.9	5.9	46.7
New: SA	6.5	99.1	3.8	91.4	3.4	90.3	4.8	90.8	6.4	91.6
New: NA	58.6	100	28.1	100	11.4	99.8	11.4	100	9.5	100
New: POA	74.0	100	32.4	100	31.3	100	13.6	100	12.5	100
Full search	1300	100	2450	100	7300	100	11,700	100	19,400	100

The processing times of the SA, NA and POA include the time required for generating the reference approximation by D–P.

6. Conclusions

The proposed iterative reduced-search algorithm consists of the following three steps: (1) generating a rough approximation by any fast heuristic algorithm, (2) constructing a bounding corridor in the state space along the reference solution, and (3) searching the minimum cost path in the corridor using dynamic programming. The algorithm can be iterated using the output solution as a new reference path in the next iteration.

The time complexity of the algorithm is proportional to W^2N^2/M, which varies from $O(N)$ to $O(N^2)$ depending on the number of output segments M. The trade-off between the run time and fidelity can be regulated in a wide range by the selection of the corridor width W, and the number of iterations. The space complexity of the algorithm is proportional to NW.

To sum up, the algorithm is not sensitive to the choice of the initial approximation, and it can be used as an additional optimization step to improve the result of any other sub-optimal approximation method. We can conclude that the algorithm bridges the gap between the optimal but slow algorithms, and the fast sub-optimal heuristic algorithms. The algorithm can be used for obtaining optimal results by setting-up the corridor width large enough. With a narrow corridor, on the other hand, the algorithm serves as a fast near-optimal heuristic.

References

Chan, W.S., Chin, F., 1996. On approximation of polygonal curves with minimum number of line segments or minimum error. Int. J. Comput. Geometry Appl. 6, 59–77.

Chen, D.Z., Daescu, O., 1998. Space-efficient algorithms for approximating polygonal curves in two-dimensional space, in: Proc. of the 4th Ann. Int. Conf. on Computing and Combinatorics, Taipei, Lecture Notes in Computer Science, 1449, 45–54, Springer, Berlin.

Douglas, D.H., Peucker, T.K., 1973. Algorithm for the reduction of the number of points required to represent a line or its caricature. The Canadian Cartographer 10 (2), 112–122.

Fränti, P., Ageenko, E., Kolesnikov, A., 1999. Vectorizing and feature-based filtering for line-drawing image compression. Pattern Anal. Appl. 2, 285–291.

Huang, S.-C., Sun, Y.-N., 1999. Polygonal approximation using genetic algorithms. Pattern Recognit. 32, 1017–1026.

Imai, H., Iri, M., 1986. Computational-geometric methods for polygonal approximations of a curve. Comput. Vision Image Process. 36, 31–41.

Imai, H., Iri, M., 1988. Polygonal approximations of a curve (formulations and algorithms). In: Toussaint, G.T. (Ed.), Computational Morphology. North-Holland, Amsterdam, pp. 71–86.

Katsaggelos, A.K., Kondi, L.P., Meier, F.W., Osterman, J., Schuster, G.M., 1998. MPEG-4 and rate-distortion-based shape-coding techniques. Proc. IEEE 86 (6), 1126–1154.

Melkman, A., O'Rourke, J., 1988. On polygonal chain approximation. In: Toussaint, G.T. (Ed.), Computational Morphology. North-Holland, Amsterdam, pp. 87–95.

Perez, J.C., Vidal, E., 1994. Optimum polygonal approximation of digitized curves. Pattern Recognit. Lett. 15, 743–750.

Pikaz, A., Dinstein, I., 1995. An algorithm for polygonal approximation based on iterative point elimination. Pattern Recognit. Lett. 16, 557–563.

Ray, B.K., Ray, K.S., 1994. A non-parametric sequential method for polygonal approximation of digital curves. Pattern Recognit. Lett. 15, 161–167.

Rosin, R.L., 1997. Techniques for assessing polygonal approximations of curves. IEEE Trans. Pattern Anal. Machine Intell. 14, 659–666.

Salotti, M., 2000. Improvement of Perez and Vidal algorithm for the decomposition of digitized curves into line segments, in: Proc. of the 15th Int. Conference on Pattern Recognition 2, 882–886.

Salotti, M., 2001. An efficient algorithm for the optimal polygonal approximation of digitized curves. Pattern Recognit. Lett. 22, 215–221.

Salotti, M., 2002. Un algorithme efficiace pour l'approximation polygonale optimale. 13ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle, 8–10 Janvier.

Schuster, G.M., Katsaggelos, A.K., 1998. An optimal polygonal boundary encoding scheme in the rate distortion sense. IEEE Trans. Image Process. 7 (1), 13–26.

Wenyin, L., Dori, D., 1999. From raster to vectors: extracting visual information from line drawings. Pattern Anal. Appl. 2, 10–21.

Yin, P.-Y., 1998. Algorithms for straight line fitting using *k*-means. Pattern Recognit. Lett. 19, 31–41.

Zhang, H., Guo, J., 2001. Optimal polygonal approximation of digital planar curves using meta-heuristics. Pattern Recognit. 34, 1429–1436.

Zhu, Y., Seneviratne, L.D., 1997. Optimal polygonal approximation of digitized curves. IEEE Proc.-Vis. Image Signal Process. 144 (1), 8–14.