Piecewise linear skeletonization using principal curves *

Balázs Kégl Adam Krzyżak

IEEE Transactions on Pattern Analysis and Machine Intelligence
vol. 24, no. 1, pp. 59-74, 2002.

Abstract

We propose an algorithm to find piecewise linear skeletons of hand-written characters by using principal curves. The development of the method was inspired by the apparent similarity between the definition of principal curves (smooth curves which pass through the "middle" of a cloud of points) and the medial axis (smooth curves that go equidistantly from the contours of a character image). The central fitting-and-smoothing step of the algorithm is an extension of the polygonal line algorithm [1, 2] which approximates principal curves of data sets by piecewise linear curves. The polygonal line algorithm is extended to find principal graphs and complemented with two steps specific to the task of skeletonization: an initialization method to capture the approximate topology of the character, and a collection of restructuring operations to improve the structural quality of the skeleton produced by the initialization method. An advantage of our approach over existing methods is that we optimize the skeleton graph by minimizing an intuitive and explicit objective function that captures the two competing criteria of smoothing the skeleton and fitting it closely to the pixels of the character image. We tested the algorithm on isolated hand-written digits and images of continuous handwriting. The results indicate that the proposed algorithm finds a smooth medial axis of the great majority of a wide variety of character templates, and substantially improves the pixelwise skeleton obtained by traditional thinning methods.

Key words: skeletonization, principal curves, feature extraction, image processing.

*B. Kégl is with the Department of Computer Science and Operational Research, University of Montreal, C.P. 6128 Succ. Centre-Ville, Canada, H3C 3J7 (email: kegl@iro.umontreal.ca). A. Krzyżak is with the Department of Computer Science, Concordia University, 1450 de Maisonneuve Blvd. West, Montreal PQ, Canada H3G 1M8 (email: krzyzak@cs.concordia.ca).

1 Introduction

Skeletonization is one of the important areas in image processing. It is most often, although not exclusively, used for images of hand-written or printed characters so we describe it here in this context. When we look at the image of a letter, we see it as a collection of curves rather than a raster of pixels. Since the earliest days of computers, it has been one of the challenges for researchers working in the area of pattern recognition to imitate this ability of the human mind [3, 4]. Approaching skeletonization from a practical point of view, representing a character by a set of thin curves rather than by a raster of pixels is useful for reducing the storage space and processing time of the character image. It was found that this representation is particularly effective in finding relevant features of the character for optical character recognition [5, 6].

The objective of skeletonization is to find the medial axis of a character. Ideally, the medial axis is defined as a smooth curve (or set of curves) that follows the shape of a character equidistantly from its contours [7]. In case of hand-written characters, one can also define the medial axis as the trajectory of the penstroke that created the letter. Most skeletonization algorithms approximate the medial axis by a unit-width binary image. In one of the most widely-used strategies, this binary image is obtained from the original character by iteratively peeling its contour pixels until there remains no more removable pixel [8, 9, 10]. The process is called *thinning*, and the result is the *skeleton* of the character. The different thinning methods are characterized by the rules that govern the deletion of black pixels.

In this paper we propose another approach to skeletonization. The development of the method was inspired by the apparent similarity between the definition of principal curves and the medial axis. Principal curves were defined by Hastie and Stuetzle [11, 12] (hereafter HS) as "self consistent" smooth curves which pass through the "middle" of a d-dimensional probability distribution or data cloud, whereas the medial axis is a set of smooth curves that go equidistantly from the contours of a character. Therefore, by representing the black pixels of a character by a two-dimensional data set, one can use the principal curve of the data set to approximate the medial axis of the character.

Kégl et al. [1, 2] redefined principal curves and proposed a practical algorithm for estimating the principal curve of a data set. The basic idea of the polygonal line (PL) algorithm is to start with a straight line segment and in each iteration of the algorithm to increase the number of segments by one by adding a new vertex to the polygonal curve produced in the previous iteration. After adding a new vertex, the positions of all vertices are updated in an inner loop so that the resulting curve minimizes a penalized distance function. The evolution of the curve produced by the algorithm is illustrated by an example in Figure 1. [1] demonstrated that the PL algorithm compares favorably with the algorithm proposed by HS both in terms of performance and computational complexity,

and is more robust to varying data models.

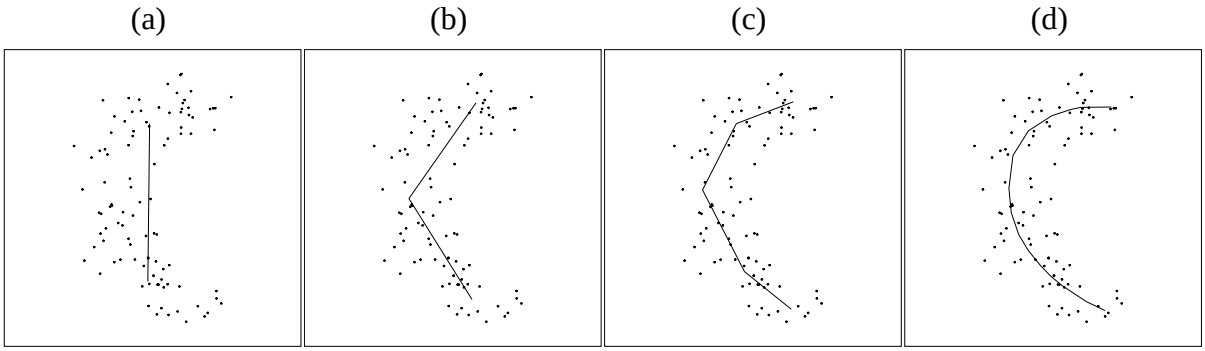


Figure 1: The curves produced by the PL algorithm for 100 data points after the (a) first, (b) second, (c) fourth, and (d) fifteenth iteration (the output of the algorithm). The data was generated by adding independent Gaussian errors to both coordinates of a point chosen randomly on a half circle.

The PL algorithm was tested on images of isolated handwritten digits from the NIST Special Database 19 [13]. We found that the PL algorithm can be used effectively to find smooth medial axes of simple digits which contain no loops or crossings of strokes. Figure 2 shows some of these results. Since the medial axis can be a *set* of connected curves rather then only *one* curve, in this paper we extend the PL algorithm to find a *principal graph* of a data set. The extended algorithm also contains two elements specific to the task of skeletonization, an initialization method to capture the approximate topology of the character, and a collection of restructuring operations to improve the structural quality of the skeleton produced by the initialization method. To avoid confusion, in what follows we use the term skeleton for the unit-width binary image approximating the medial axis, and we refer to the set of connected curves produced by the PL algorithm as the *skeleton graph* of the character template.

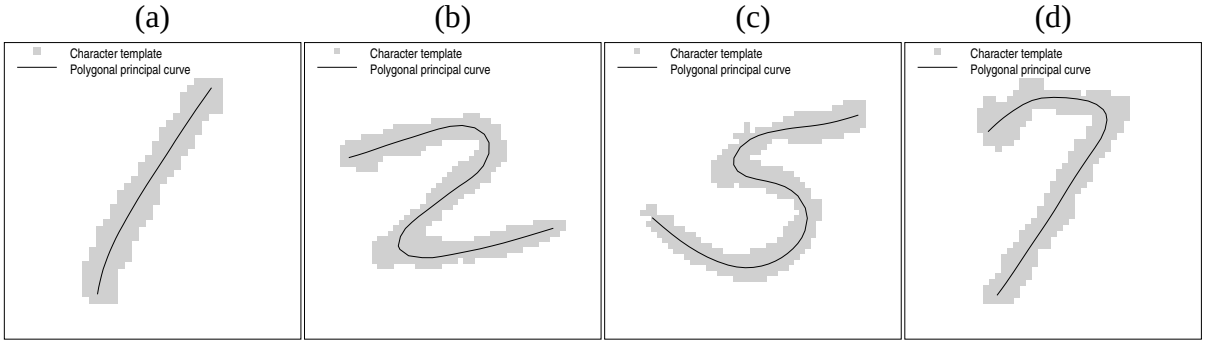


Figure 2: The PL algorithm can be used effectively to find smooth medial axes of simple digits which contain no loops or crossings of strokes.

Principal curves have been previously used in the area of image processing by Banfield and Raftery [14] and Singh et al. [15]. [14] described an almost fully automatic method for identi-

fying ice floes and their outlines in satellite images. The main procedure of the method uses a closed principal curve to estimate the floe outlines. The authors eliminated the "flattening" estimation bias of the HS algorithm and also replaced the initialization step of the HS algorithm by a more sophisticated routine that produced a rough estimate of the floe outlines. Furthermore, [14] extended existing clustering methods by allowing groups of data points to be centered about principal curves rather than points or lines. [15] used the HS algorithm for skeletonization. The initial curve is produced by a variant of Kohonen's self-organizing map (SOM) algorithm [16] where the neighborhood relationships are defined by a minimum spanning tree of the pixels of the character template. The HS algorithm is then used to fit the curve to the character template. In the expectation step, a weighted kernel smoother is used which, in this case, is equivalent to the update rule of the SOM algorithm.

Similar skeletonization methods were proposed by Mahmoud et al. [17] and Datta and Parui [18]. Similarly to [15], [18] uses the SOM algorithm to optimize the positions of vertices of a piecewise linear skeleton. The algorithm follows a "bottom-up" strategy in building the skeletal structure: the approximation starts from a linear topology and later adds forks and loops to the skeleton based on local geometric patterns formed during the SOM optimization. [17] proposed an algorithm to obtain piecewise linear skeletons of Arabic characters. The method is based on fuzzy clustering and the fuzzy ISODATA algorithm [19] that uses a similar optimization to the batch version of the SOM algorithm.

The rest of the paper is organized as follows. In Section 2, we describe the extended PL algorithm to find the principal graph of a data set. In Section 3 test results of the principal graph algorithm are presented. In Section 3.1 we apply the algorithm to isolated hand-written digits from the NIST Special Database 19 [13]. The results indicate that the proposed algorithm finds a smooth medial axis of the great majority of a wide variety of character templates, and substantially improves the pixelwise skeleton obtained by traditional thinning methods. In Section 3.2 we present results of experiments with images of continuous handwriting. These experiments demonstrate that the skeleton graph produced by the algorithm can be used for representing hand-written text efficiently. In Section 4, we conclude our work by discussing the relationship between the principal graph algorithm and the methods proposed in [17, 18, 15], and outlining the directions of future research.

2 The Principal Graph Algorithm

The objective of the principal graph algorithm[1] is to fit a set of smooth, piecewise linear curves to the image of the character. The curves are connected to each other at certain vertices, so they form a Euclidean graph in the plane spanned by the midpoints of the pixels of the image. A Euclidean graph $G_{\mathcal{V},\mathcal{S}}$ in the d-dimensional Euclidean space is defined by two sets, $\mathcal{V}$ and $\mathcal{S}$, where $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\} \subset \mathbb{R}^d$ is a set of *vertices*, and $\mathcal{S} = \{(\mathbf{v}_{i_1}, \mathbf{v}_{j_1}), \ldots, (\mathbf{v}_{i_k}, \mathbf{v}_{j_k})\} = \{\mathbf{s}_{i_1 j_1}, \ldots, \mathbf{s}_{i_k, j_k}\}$, $1 \leq i_1, j_1, \ldots, i_k, j_k \leq m$ is a set of *edges*, such that $\mathbf{s}_{ij}$ is a line segment that connects $\mathbf{v}_i$ and $\mathbf{v}_j$. For the sake of simplicity, when it does not cause confusion, we shall use a single index to enumerate the edges of $G_{\mathcal{V},\mathcal{S}}$, i.e., we shall write $\mathcal{S} = \{\mathbf{s}_i, \ldots, \mathbf{s}_k\}$. Throughout the paper we shall use the term *graph* as an abbreviation for Euclidean graph. We also omit the indices of $G_{\mathcal{V},\mathcal{S}}$ if it does not cause confusion.

The principal graph algorithm starts by an initializing routine based on a traditional thinning method (Section 2.2). The initial graph captures the approximate topology of the character, and it roughly follows the medial axis of the character. However, it is not smooth and it usually contains a number of spurious branches and inadequate structural elements. To solve these two problems, the principal graph algorithm proceeds by a restructuring step sandwiched between two fitting-and-smoothing steps (Figure 3). In the restructuring step (Section 2.3), we apply several operations to rectify the structural imperfections of the skeleton graph. Figure 4 illustrates the evolution of the skeleton graph on an example.

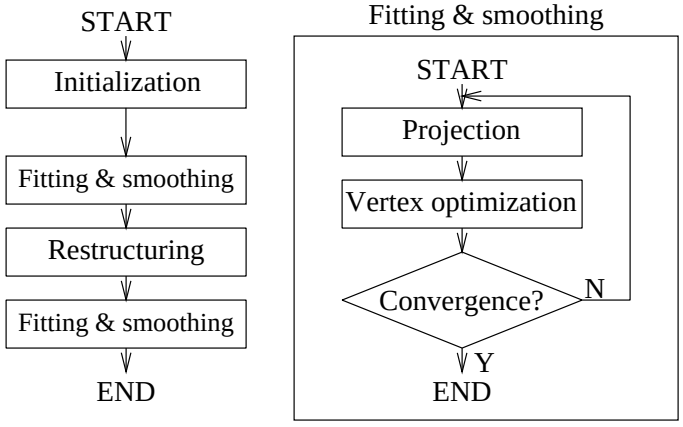


Figure 3: The flow-chart of the principal graph algorithm.

The main component of the algorithm is the fitting-and-smoothing routine (Section 2.1). In this

[1]The Java implemantation of the algorithm is available at the WWW site

`http://www.iro.umontreal.ca/~kegl/skeletonization.html`

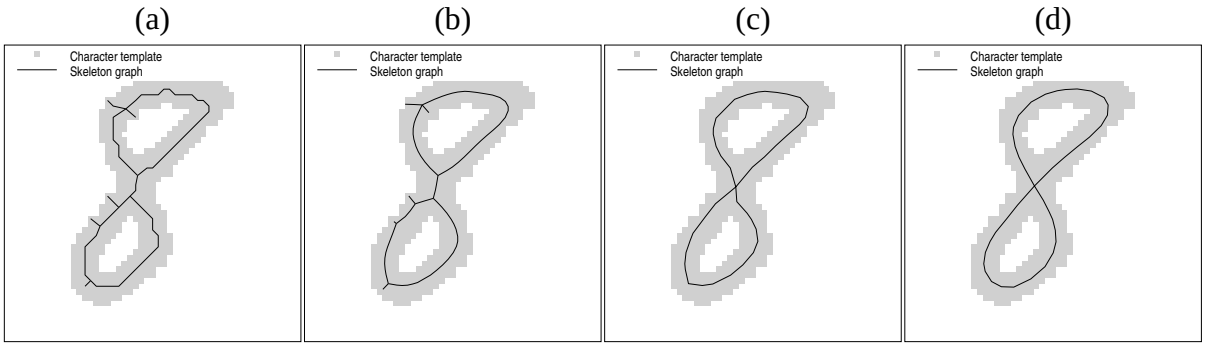


Figure 4: Evolution of the skeleton graph. The skeleton graph produced by the principal graph algorithm (a) after the initialization step, (b) after the first fitting-and-smoothing step, (c) after the restructuring step, and (d) after the second fitting-and-smoothing step (the output of the algorithm).

step we use an extended version of the PL algorithm to iteratively smooth the skeleton graph while keeping it approximately equidistant from the contours of the character. The method is based on the minimization of a simple and intuitive "energy function", and it requires almost no "engineering" if applied to images of different nature or combined with other methods. On the other hand, the initialization and restructuring steps depend heavily on the type of input and on each other, and they require the manual fine-tuning of several threshold parameters. They are, however, independent of the fitting-and-smoothing step, so they can be easily replaced if necessary. Novel, more principled skeletonization methods, such as veinerization [20] or the stochastic jump-diffusion algorithm of Zhu [21] can play a principal role here.

In principle, the restructuring step could even be eliminated: if the fitting-and-smoothing module is given a good initial skeleton, one fitting-and-smoothing step is enough. The main justification of our algorithmic structure is that we found that we can acquire much more high-level structural information about the skeleton *after* it has been smoothed once which, combined with some a-priori knowledge about the input data, can be used to reduce noise and to correct structural imperfections.

2.1 The Fitting-And-Smoothing Step

This step plays the central role of smoothing the skeleton graph and fitting it to the character. We describe it before other steps not only because it is the main component of the algorithm but also because we define several concepts here that will be used in the description of the initialization and the restructuring steps.

In the fitting-and-smoothing step, given a data set[2] $\mathcal{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$, we attempt to

[2]To transform binary (black-and-white) character templates into two-dimensional data sets, we place the midpoint of the bottom-most left-most pixel of the template to the center of a coordinate system. The unit length of the coordi-

optimize the skeleton graph G by minimizing a penalized distance function $E(G)$ defined as

$$E(G) = \Delta(G) + \lambda P(G). \tag{1}$$

The first component $\Delta(G)$ is the *average squared distance* of points in $\mathcal{X}_n$ from the graph G defined by

$$\Delta(G) = \frac{1}{n}\sum_{i=1}^{n} \Delta(\mathbf{x}_i, G)$$

where $\Delta(\mathbf{x}_i, G)$ is the Euclidean squared distance of a point $\mathbf{x}_i$ and the nearest point of the graph G to $\mathbf{x}_i$. The second component $P(G)$ is a penalty on the total curvature of the graph defined by

$$P(G) = \frac{1}{m}\sum_{i=1}^{m} P_{\mathbf{v}}(\mathbf{v}_i) \tag{2}$$

where m is the number of vertices of G and $P_{\mathbf{v}}(\mathbf{v}_i)$ is the curvature penalty at vertex $\mathbf{v}_i$. In general, $P_{\mathbf{v}}(\mathbf{v}_i)$ is small if edges incident to $\mathbf{v}_i$ join smoothly at $\mathbf{v}_i$. An important general property of $P_{\mathbf{v}}(\mathbf{v}_i)$ that it is local in the sense that it can change only if $\mathbf{v}_i$ or adjacent vertices to $\mathbf{v}_i$ are relocated. The exact form of $P_{\mathbf{v}}(\mathbf{v}_i)$ is presented in Section 2.1.1 where vertices of different types are described.

Achieving a low average distance means that the skeleton graph closely fits the data. Keeping $P(G)$ low ensures the smoothness of the skeleton graph. The penalty coefficient λ plays the balancing role between these two competing criteria. Based on heuristic considerations explained below, and after carrying out practical experiments, we set

$$\lambda = 0.1 \cdot \frac{m}{n^{1/3}} \cdot \frac{\sqrt{\Delta(G)}}{r}$$

where r is the "radius" of the data defined by $r = \max_{\mathbf{x}\in\mathcal{X}_n} \left\|\mathbf{x} - \frac{1}{n}\sum_{\mathbf{y}\in\mathcal{X}_n}\mathbf{y}\right\|$. By setting the penalty to be proportional to the average distance of the data points from the graph (normalized by the radius) and to the number of vertices in the skeleton graph, we avoid the zig-zagging behavior of the graph resulting from overfitting when the character is relatively thick. At the same time, this penalty formulation allows the principal graph to closely follow the shape of the character when the character is relatively thin. The exact form of the factor $\frac{m}{n^{1/3}}$ derives from a theoretical consideration explained in [1]. The constant factor 0.1 was determined experimentally.

$E(G)$ is a complicated nonlinear function of G so finding its minimum analytically is impossible. Furthermore, simple gradient-based optimization methods also fail since $E(G)$ is not differentiable at certain points. To minimize $E(G)$, we iterate between a projection step and a vertex optimization step until convergence (Figure 3). In the projection step, the data points are partitioned into "nearest neighbor regions" according to which edge or vertex they project. The

nate system is set to the width (and height) of a pixel, so the midpoint of each pixel has integer coordinates. Then we add the midpoint of each black pixel to the data set.

resulting partition is formally defined in Section 2.1.2 and illustrated in Figure 5. In the vertex optimization step (Section 2.1.3), we use a gradient-based method to minimize $E(G)$ assuming that the partition computed in the previous projection step does not change. Under this condition the objective function becomes differentiable everywhere so a gradient-based method can be used for finding a local minimum. The drawback is that if the assumption fails to hold, that is, some data points leave their nearest neighbor regions while vertices of the graph are moved, the "real" objective function $E(G)$ might increase in this step. As a consequence, the convergence of the fitting-and-smoothing iteration cannot be guaranteed in theory. In practice, during extensive test runs, however, the algorithm was observed to always converge.

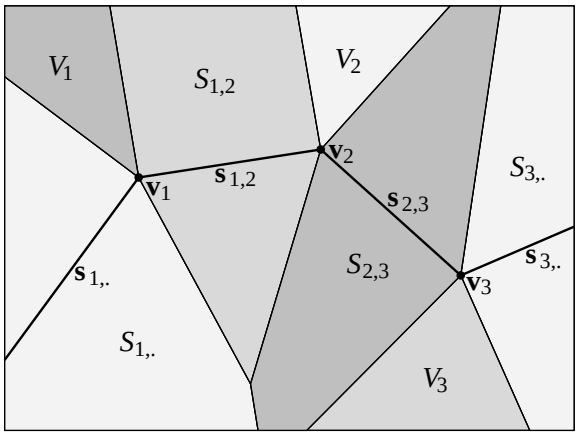


Figure 5: A nearest neighbor partition of $\mathbb{R}^2$ induced by the vertices and edges of a graph. The nearest point of the graph to any point in the set V_i is the vertex $\mathbf{v}_i$. The nearest point of the graph to any point in the set S_{ij} is an *inner* point of the line segment $\mathbf{s}_{ij}$.

Several other unsupervised learning methods contain similar optimization routines to our fitting-and-smoothing iteration. The Generalized Lloyd (GL) algorithm [22] (also known as the *k*-means algorithm) of vector quantizer design alternates between an expectation and a partition step until convergence. In the partition step the nearest neighbor sets of the codepoints are set, and in the expectation step the average squared distance is minimized by setting the new codepoints to the center of gravity of the nearest neighbor sets. In the batch version of the SOM algorithm [23, 16] the partition step of the GL algorithm is coupled with a more sophisticated expectation step where codepoints are set to weighted averages of the data points. A similar iteration is used in a recently proposed method for computing principal components of data sets [24, 25]. This method alternates between projecting the data to a linear subspace and optimizing the average squared distance of the data from the subspace assuming the fixed projection indexes. The principal graph algorithm is also related in spirit to the expectation-minimization (EM) algorithm [26] although we can show no probabilistic model for which it is actually an EM algorithm. Note that a more complete

convergence analysis and extensive comparison to other unsupervised methods is given in [2].

2.1.1 Vertex Types and the Curvature Penalty

The PL algorithm differentiates between vertices at the end and in the middle of the polygonal curve in defining the curvature penalty $P_{\mathbf{v}}(\mathbf{v}_i)$. We call these vertices *end-vertices* and *line-vertices*, respectively. At a line-vertex, the smoothness of the polygonal curve is ensured by penalizing the angle of the two incident edges for its deviation from straight angle. At an end-vertex, the penalty on a nonexistent angle is replaced by a penalty on the squared length of the incident edge. In this section we introduce new vertex types to accommodate intersecting curves that occur in handwritten characters. Vertices of different types are characterized by their degrees and the types of the curvature penalty imposed at them (Table 1).

The only vertex type of degree one is the end-vertex. If two edges are joined by a vertex, the vertex is either a line-vertex or a *corner-vertex*. At a corner vertex we penalize the angle for its deviation from right angle. We introduce three different vertex types of degree three. At a *star3-vertex*, no penalty is imposed. At a *T-vertex*, we penalize one of the three angles for its deviation from straight angle. The remaining two angles are penalized for their deviations from right angle. At a *Y-vertex*, two of the possible angles are penalized for their deviations from straight angle. We use only two of the several possible configurations at a vertex of degree four. At a *star4-vertex* no penalty is imposed, while at an *X-vertex* we penalize sharp angles on the two crossing curves. Vertices of degree three or more are called *junction vertices*.

In principle, several other types of vertices can be considered. However, in practice we found that these types are sufficient to represent hand-written characters from the Latin alphabet and of the ten digits. Vertices at the end and in the middle of a curve are represented by end-vertices and line-vertices, respectively. Two curves can be joined at their endpoints by a corner-vertex (Figure 6(a)). The role of a Y-vertex is to "merge" two smooth curves into one (Figure 6(b)). A T-vertex is used to join the end of a curve to the middle of another curve (Figure 6(c)). An X-vertex represents the crossing point of two smooth curves (Figure 6(d)). Star3 and star4-vertices are used in the first fitting-and-smoothing step, before we make the decision on the penalty configuration at a particular junction-vertex.

For the definition of the curvature penalty, we first introduce $\pi_{ij\ell} = r^2(1+\cos\gamma_{ij\ell})$, where $\gamma_{ij\ell}$ denotes the angle of edges $\mathbf{s}_{ji}$ and $\mathbf{s}_{j\ell}$, to penalize the deviation of $\gamma_{ij\ell}$ from straight angle. The factor r^2 is used to ensure scale independence. At end-vertices, we use $\mu_{ij} = \|\mathbf{v}_i - \mathbf{v}_j\|^2$ to penalize the length of the incident edge $\mathbf{s}_{ij}$. At corner and T-vertices we introduce $\omega_{ij\ell} = 2r^2\cos^2\gamma_{ij\ell}$ to penalize the deviation of $\gamma_{ij\ell}$ from right angle. The penalties are formally defined in Table 1 for the different vertex types.

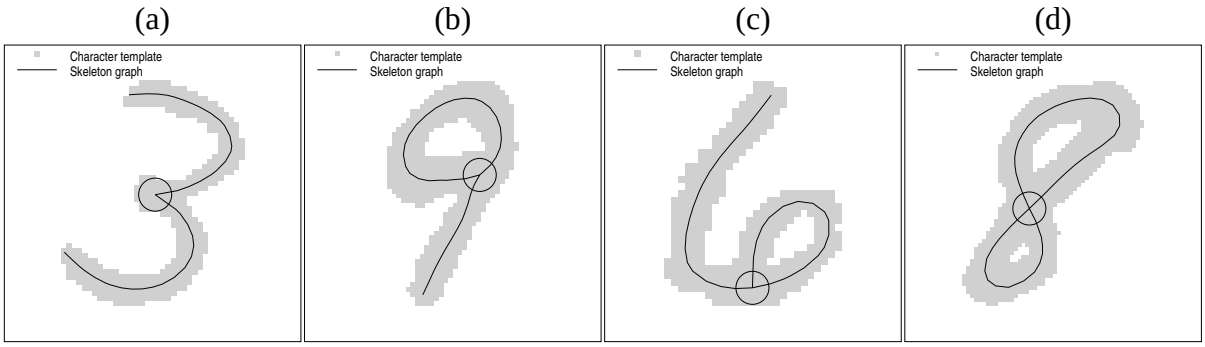


Figure 6: Roles of vertices of different types. (a) A corner-vertex joins two curves at their endpoints. (b) A Y-vertex merges two smooth curves into one. (c) A T-vertex joins the end of a curve to the middle of another curve. (d) An X-vertex represents the crossing point of two smooth curves.

Type of $\mathbf{v}_i$	$\phi(\mathbf{v}_i)$	Penalty at $\mathbf{v}_i$	Configuration
end	1	$P_{\mathbf{v}}(\mathbf{v}_i) = \mu_{i,i_1}$	
line	2	$P_{\mathbf{v}}(\mathbf{v}_i) = \pi_{i_1,i,i_2}$	
corner	2	$P_{\mathbf{v}}(\mathbf{v}_i) = \omega_{i_1,i,i_2}$	
star3	3	$P_{\mathbf{v}}(\mathbf{v}_i) = 0$	
T	3	$P_{\mathbf{v}}(\mathbf{v}_i) = \pi_{i_2,i,i_3} + \omega_{i_1,i,i_2} + \omega_{i_1,i,i_3}$	
Y	3	$P_{\mathbf{v}}(\mathbf{v}_i) = \pi_{i_1,i,i_2} + \pi_{i_1,i,i_3}$	
star4	4	$P_{\mathbf{v}}(\mathbf{v}_i) = 0$	
X	4	$P_{\mathbf{v}}(\mathbf{v}_i) = \pi_{i_1,i,i_4} + \pi_{i_2,i,i_3}$	

Table 1: Vertex types and their attributes. The third column defines the penalties applied at each vertex type. The arcs in the fourth column indicate the penalized angles. The solid arc and the dashed arc indicate that the angle is penalized for its deviation from straight angle and from right angle, respectively.

Most of the restructuring operations to be described in Section 2.3 proceed by deleting noisy edges and vertices from the skeleton graph. When an edge is deleted from the graph, the degrees of the two incident vertices decrease by one. Since there exist more than one vertex types for a given degree, the new types of the degraded vertices must be explicitly specified by degradation rules. When an edge incident to an end-vertex is deleted, we delete the vertex to avoid singular points in the skeleton graph. Line and corner-vertices are degraded to end-vertices, while star4-vertices are degraded to star3-vertices. Any vertex of degree three is degraded to a line-vertex if the remaining angle was penalized for its deviation from straight angle before the degradation, or if the angle is larger than 100 degrees. Otherwise, it is degraded to a corner-vertex. An X-vertex is degraded to

a T-vertex if both of the two unpenalized angles are between 80 and 100 degrees, otherwise it is degraded to a Y-vertex. The explicit degradation rules are given in Table 2.

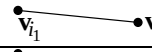
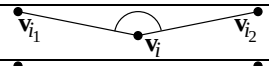
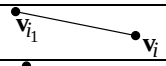
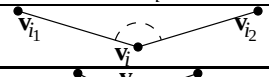
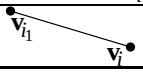
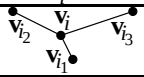
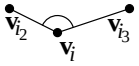
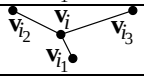
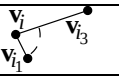
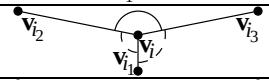
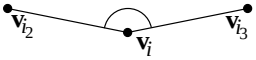
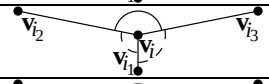
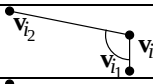
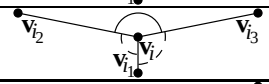
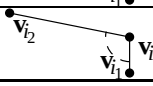
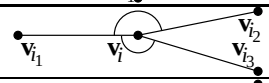
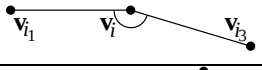
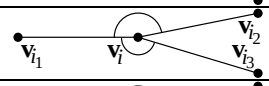
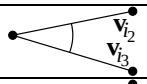
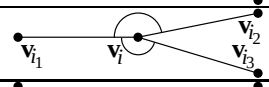
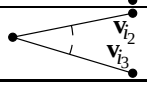
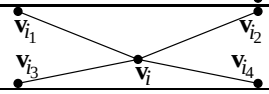
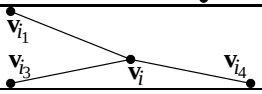
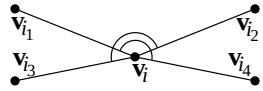
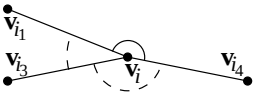
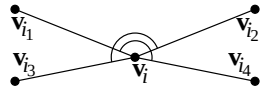
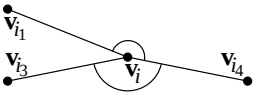
Type (before)	Configuration (before)	Deleted edge	Type (after)	Configuration (after)	Conditions
end		$\mathbf{s}_{i,i_1}$	*deleted*	–	–
line		$\mathbf{s}_{i,i_2}$	end		–
corner		$\mathbf{s}_{i,i_2}$	end		–
star3		$\mathbf{s}_{i,i_1}$	line		$\gamma_{i_2,i,i_3} > 100°$
star3		$\mathbf{s}_{i,i_2}$	corner		$\gamma_{i_1,i,i_3} \leq 100°$
T		$\mathbf{s}_{i,i_1}$	line		–
T		$\mathbf{s}_{i,i_3}$	line		$\gamma_{i_1,i,i_2} > 100°$
T		$\mathbf{s}_{i,i_3}$	corner		$\gamma_{i_1,i,i_2} \leq 100°$
Y		$\mathbf{s}_{i,i_2}$	line		–
Y		$\mathbf{s}_{i,i_1}$	line		$\gamma_{i_2,i,i_3} > 100°$
Y		$\mathbf{s}_{i,i_1}$	corner		$\gamma_{i_2,i,i_3} \leq 100°$
star4		$\mathbf{s}_{i,i_2}$	star3		–
X		$\mathbf{s}_{i,i_2}$	T		$80° \leq \gamma_{i_1,i,i_3} \leq 100°$, $80° \leq \gamma_{i_3,i,i_4} \leq 100°$
X		$\mathbf{s}_{i,i_2}$	Y		not as above, $\gamma_{i_1,i,i_4} > \gamma_{i_1,i,i_3}$, $\gamma_{i_3,i,i_4} > \gamma_{i_1,i,i_3}$

Table 2: Vertex degradation rules.

2.1.2 The Projection Step

The objective of this step to recompute the nearest neighbor (Voronoi) partition induced by the vertices and edges of the skeleton graph indicated by Figure 5. After the step, each data point $\mathbf{x}$ will belong to the set V_i if the nearest point of the graph to $\mathbf{x}$ is the vertex $\mathbf{v}_i$, and it will belong to the set S_i if the nearest point of the graph to $\mathbf{x}$ is an *inner* point of the line segment S_i.

Formally, let $G_{\mathcal{V},\mathcal{S}}$ be a Euclidean graph with m vertices and k edges, and let $\mathcal{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset$

$\mathbb{R}^d$ be a set of data points. For each vertex $\mathbf{v}_i \in \mathcal{V}$, $i = 1,\ldots,m$ and edge $\mathbf{s}_i \in \mathcal{S}$, $i = 1,\ldots,k$ we define the nearest neighbor sets V_i and S_i, respectively, by

$$V_i = \left\{\mathbf{x} \in \mathcal{X}_n : \Delta(\mathbf{x},\mathbf{v}_i) = \Delta(\mathbf{x}, G_{\mathcal{V},\mathcal{S}}),\ \Delta(\mathbf{x},\mathbf{v}_i) < \Delta(\mathbf{x},\mathbf{v}_j),\ j = 1,\ldots,i-1\right\}$$

and

$$S_i = \left\{\mathbf{x} \in \mathcal{X}_n : \mathbf{x} \notin \bigcup_{j=1}^{m} V_j, \Delta(\mathbf{x},\mathbf{s}_i) = \Delta(\mathbf{x}, G_{\mathcal{V},\mathcal{S}}), \Delta(\mathbf{x},\mathbf{s}_i) < \Delta(\mathbf{x},\mathbf{s}_j), j = 1,\ldots,i-1\right\}$$

where $\Delta(\mathbf{x},\mathbf{v}_i) = \|\mathbf{x}-\mathbf{v}_i\|^2$, $\Delta(\mathbf{x},\mathbf{s}_i)$ is the Euclidean squared distance between a point $\mathbf{x}$ and a line segment $\mathbf{s}_i$ and $\Delta(\mathbf{x}, G_{\mathcal{V},\mathcal{S}}) = \min_{\mathbf{s}\in\mathcal{S}} \Delta(\mathbf{x},\mathbf{s})$ is the Euclidean squared distance between $\mathbf{x}$ and the graph $G_{\mathcal{V},\mathcal{S}}$.

When partitioning the data set $\mathcal{X}_n$, we first find the nearest segment $\mathbf{s}$ to each data point $\mathbf{x} \in \mathcal{X}_n$, then place $\mathbf{x}$ to either the nearest neighbor set of one of the endpoints of $\mathbf{s}$ ($\mathbf{x}$ projects to one of the endpoints) or to the nearest neighbor set of $\mathbf{s}$ itself ($\mathbf{x}$ projects to an inner point of $\mathbf{s}$). Note that the somewhat simpler procedure of finding the nearest vertex $\mathbf{v}$ to a data point $\mathbf{x}$ and then placing $\mathbf{x}$ into the nearest neighbor set of either $\mathbf{v}$ or one of the incident segments of $\mathbf{v}$ can result in an incorrect partition since in general the nearest segment to $\mathbf{x}$ does not have to be incident to the nearest vertex of $\mathbf{x}$.

2.1.3 The Vertex Optimization Step

During the optimization step we assume that none of the data points leave the nearest neighbor cell of an edge or a vertex. This is clearly an incorrect assumption but without it it would be hard to use a gradient-based minimization method since the distance of a point $\mathbf{x}$ and the skeleton graph is not differentiable (with respect to the vertices of the graph) if $\mathbf{x}$ falls on the boundary of two nearest neighbor regions. Also, to check whether a data point has left the nearest neighbor cell of an edge or a vertex, we would have to execute a projection step each time when a vertex is moved, which is computationally inefficient. On the other hand, the gradient can be computed very efficiently (in O(1) time) if the partition is fixed.

Technically, the assumption of a fixed partition means that the distance of a data point $\mathbf{x}$ and a line segment $\mathbf{s}_{ij}$ is measured as if $\mathbf{s}_{ij}$ were an infinite line. Accordingly, let $\mathbf{s}'_{ij}$ be the line obtained by the infinite extension of the line segment $\mathbf{s}_{ij}$, and let

$$\sigma(\mathbf{s}_{ij}) = \sum_{\mathbf{x}\in S_{ij}} \Delta(\mathbf{x},\mathbf{s}'_{ij}) \quad \text{and} \quad \nu(\mathbf{v}_i) = \sum_{\mathbf{x}\in V_i} \Delta(\mathbf{x},\mathbf{v}_i)$$

where $\Delta(\mathbf{x},\mathbf{s}'_{ij})$ is the Euclidean squared distance of the point $\mathbf{x}$ and the infinite line $\mathbf{s}'_{ij}$. Note that after every projection step, before the vertices of the graph are moved, the distance function of the

graph is given by

$$\Delta(G_{\mathcal{V},\mathcal{S}}) = \frac{1}{n}\left(\sum_{\mathbf{v}\in\mathcal{V}} \nu(\mathbf{v}) + \sum_{\mathbf{s}\in\mathcal{S}} \sigma(\mathbf{s})\right).$$

The gradient of the objective function $E(G_{\mathcal{V},\mathcal{S}})$ with respect to a vertex $\mathbf{v}_i$ can be computed locally in the following sense. On the one hand, according to the assumption that data points do not cross boundaries of nearest neighbor cells, only distances of data points that project to $\mathbf{v}_i$ or incident edges to $\mathbf{v}_i$ can change when $\mathbf{v}_i$ is moved. On the other hand, when the vertex $\mathbf{v}_i$ is moved, only angles at $\mathbf{v}_i$ and at neighbors of $\mathbf{v}_i$ can change. Therefore, the gradient of $E(G)$ with respect to $\mathbf{v}_i$ can be computed as

$$\nabla_{\mathbf{v}_i} E(G) = \nabla_{\mathbf{v}_i}\big(\Delta(G) + \lambda P(G)\big) = \nabla_{\mathbf{v}_i}\big(\Delta(\mathbf{v}_i) + \lambda P(\mathbf{v}_i)\big)$$

where

$$\Delta(\mathbf{v}_i) = \frac{1}{n}\left(\nu(\mathbf{v}_i) + \sum_{j=1}^{\phi_i} \sigma(\mathbf{s}_{i,i_j})\right) \quad \text{and} \quad P(\mathbf{v}_i) = \frac{1}{m}\left(P_{\mathbf{v}}(\mathbf{v}_i) + \sum_{j=1}^{\phi_i} P_{\mathbf{v}}(\mathbf{v}_{i_j})\right)$$

where ϕ_i is the degree of $\mathbf{v}_i$ ($1 \le \phi_i \le 4$), and $i_1, \ldots, i_{\phi_i}$ are the indices of the adjacent vertices to $\mathbf{v}_i$. Once the gradients $\nabla_{\mathbf{v}_i} E(G)$, $i = 1, \ldots, m$ are computed, a local minimum of $E(G)$ can be obtained by any gradient-based optimization method.

2.2 The Initialization Step

The most important requirement for the initial graph is that it approximately capture the topology of the original character template. We use a traditional connectedness-preserving thinning technique that works well for moderately noisy images. If the task is to recover characters from noisy or faded images, this initialization procedure can be replaced by a more sophisticated routine (e.g., the method based on minimum spanning tree presented in [15]) without modifying other modules of the algorithm.

We selected the particular thinning algorithm based on a survey [27] which used several criteria to systematically compare twenty skeletonization algorithms. From among the algorithms that preserve connectedness, we chose the Suzuki-Abe algorithm [10] due to its high speed an simplicity. Other properties, such as reconstructability, quality of the skeleton (spurious branches, elongation or shrinkage at the end points), or similarity to a reference skeleton, were less important at this initial phase. Some of the imperfections are corrected by the fitting-and-smoothing operation, while others are treated in the restructuring step. The Suzuki-Abe algorithm starts by computing and storing the distance of each black pixel from the nearest white pixel (distance transformation). In

the second step, layers of border pixels are iteratively deleted until pixels with locally maximal distance values are reached. Finally, some of the remaining pixels are deleted so that connectedness is preserved and the skeleton is of width one.

After thinning the template, an initial skeleton graph is computed (Figure 7). In general, midpoints of pixels of the skeleton are used as vertices of the graph, and two vertices are connected by an edge if the corresponding pixels are eight-neighbors, i.e., if they have at least one common corner. This simple procedure produces graphs that may have two undesirable properties near junction pixels (pixels having more than two eight-neighbors). First, neighboring junction pixels can generate short circles in the graph. To eliminate these short circles, we find the minimum spanning tree of the subgraph created around neighboring junction pixels. Secondly, crossing strokes of the character template often generate two or more junction vertices connected to each other by short edges. Although we rectify this anomaly in the restructuring step (see Section 2.3.3), we also try to detect and eliminate easily recognizable cases here. In particular, when two or more adjacent junction vertices are detected, we delete them, and replace them with one junction vertex connected to all the non-junction neighbors of the deleted junction vertices. The new junction vertex is created in the center of gravity of its neighbors.

In this initial phase, only end, line, star3, and star4-vertices are used depending on whether vertex has one, two, three, or four neighbors, respectively. In the rare case when a vertex has more than four neighbors, the neighbors are split into two or more sets of two or three vertices by using a greedy selection based on the mutual closeness of the vertices. Each neighbor in a set is then connected to a mutual junction vertex created in the center of mess of its neighbors. The created junction vertices are connected to each other through a line-vertex (placed halfway between the two junction vertices) in order of their creation. The enlarged areas in Figures 7(c) and (d) demonstrate this case.

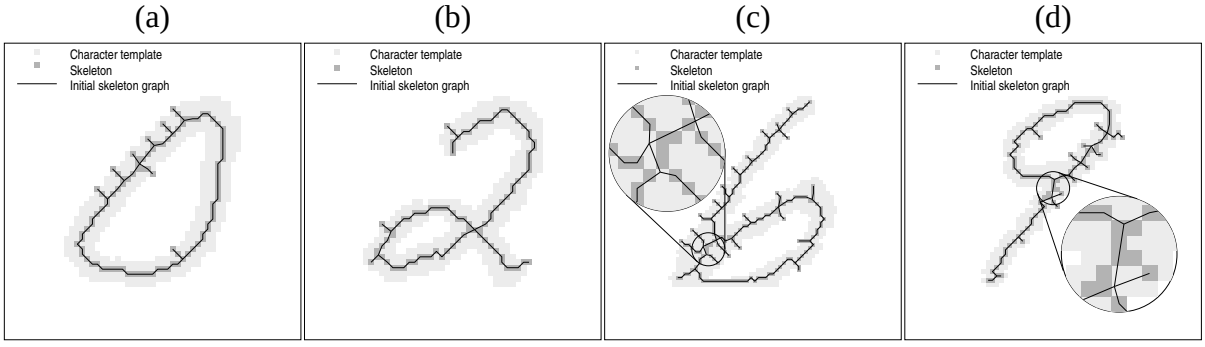


Figure 7: Examples of transforming the skeleton into an initial skeleton graph.

2.3 The Restructuring Step

The restructuring step complements the two fitting-and-smoothing steps. In the fitting-and-smoothing step we relocate vertices and edges of the skeleton graph based on their positions relative to the template, but we do not modify the skeleton graph in a graph theoretical sense. In the restructuring step we use geometric properties of the skeleton graph to modify the configuration of vertices and edges. We do not explicitly use the template, and we do not move vertices and edges of the skeleton graph in this step.

The double purpose of the restructuring step is to eliminate or rectify imperfections of the initial skeleton graph, and to simplify the skeletal description of the template. Below we define operations that can be used to modify the configuration of the skeleton graph. Since the types of the imperfections depend on properties of both the input data and the initialization method, one should carefully select the particular operations and set their parameters according to the specific application. At the description of the operations, we give approximate values for each parameter based on our experiments with a wide variety of real data. Specific settings will be given in Section 3 where we present the results of two particular experiments.

For the formal description of the restructuring operations, we define some simple concepts. We call a list of vertices $p_{i_1,\ldots,i_\ell} = (\mathbf{v}_{i_1},\ldots,\mathbf{v}_{i_\ell}), \ell > 1$ a *path* if each pair of consecutive vertices $(\mathbf{v}_{i_j},\mathbf{v}_{i_{j+1}}), j = 1,\ldots,\ell-1$ is connected by an edge. A *loop* is a path $p_{i_1,\ldots,i_\ell}$ such that $i_1 = i_\ell$ and none of the inner vertices $\mathbf{v}_{i_2},\ldots,\mathbf{v}_{i_{\ell-1}}$ are equal to each other or to $\mathbf{v}_{i_1}$. The *length* of a path is defined by

$$l(p_{i_1,\ldots,i_\ell}) = \sum_{j=1}^{\ell-1} \|\mathbf{v}_{i_{j+1}} - \mathbf{v}_{i_j}\|.$$

A path $p_{i_1,\ldots,i_\ell}$ is *simple* if its endpoints $\mathbf{v}_{i_1}$ and $\mathbf{v}_{i_\ell}$ are not line-vertices while all its inner vertices $\mathbf{v}_{i_2},\ldots,\mathbf{v}_{i_{\ell-1}}$ are line-vertices. A simple path is called a *branch* if at least one of its endpoints is an end-vertex. When we *delete* a simple path $p_{i_1,\ldots,i_\ell}$, we remove all inner vertices $\mathbf{v}_{i_2},\ldots,\mathbf{v}_{i_{\ell-1}}$ and all edges $\mathbf{s}_{i_1,i_2},\ldots,\mathbf{s}_{i_{\ell-1},i_\ell}$. Endpoints of the path $\mathbf{v}_{i_1}$ and $\mathbf{v}_{i_\ell}$ are degraded as specified by Table 2. Figure 8 illustrates these concepts.

Most of the restructuring operations simplify the skeleton graph by eliminating certain simple paths that are shorter then a threshold. To achieve scale and resolution independence, we use the *thickness* of the character as the yardstick in length measurements. We estimate the thickness of the data set $\mathcal{X}_n = \{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$ by

$$\tau = 4\sum_{i=1}^{n} \sqrt{\Delta(\mathbf{x}_i, G)}.$$

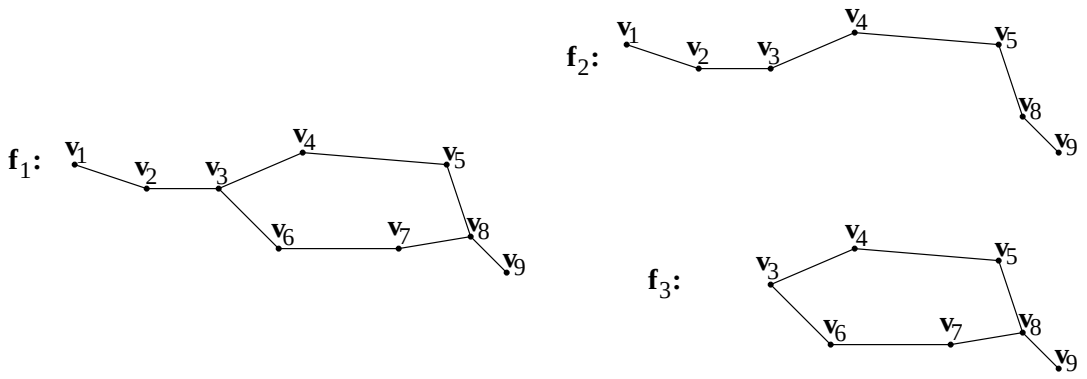


Figure 8: Paths, loops, simple paths, branches, and deletion. A loop in $\mathbf{f}_1$ is $p_{3458763}$. Simple paths of $\mathbf{f}_1$ are p_{123}, p_{3458}, p_{3678}, and p_{89}. p_{123} and p_{89} are branches of $\mathbf{f}_1$. $\mathbf{f}_2$ and $\mathbf{f}_3$ were obtained by deleting p_{3678} and p_{123}, respectively, from $\mathbf{f}_1$.

2.3.1 Deleting Short Branches

Small protrusions on the contours of the character template tend to result in short branches in the initial skeleton graph. We first approached this problem by deleting any branch that is shorter than a threshold, τ_{branch}. Unfortunately, this approach proved to be too simple in practice. By setting τ_{branch} to a relatively large value, we eliminated a lot of short branches that represented "real" parts of the character, whereas by setting τ_{branch} to a relatively small value, a lot of "noisy" branches remained in the graph. We found that after the first fitting-and-smoothing step, if the size of the protrusion is comparable to the thickness of the skeleton, i.e., the protrusion is likely to be a "real" part of the skeleton, the angles of the short branch and the connecting paths tend to be close to right angle (Figure 9(a)). On the other hand, if the short branch has been created by the noisy contour of the character, the angle of the short branch and one of the connecting path tends to be very sharp (Figure 9(b)). So, in the decision of deleting the short branch $p_{i,i_3,\ldots}$ (Figure 9), we weight the length of the branch by $w_i = 1 - \cos^2\gamma$ where $\gamma = \min(\gamma_{i_1,i,i_3}, \gamma_{i_2,i,i_3})$, and we delete $p_{i,i_3,\ldots}$ if $w_i l(p_{i,i_3,\ldots}) < \tau_{branch}$. Experiments showed that to delete most of the noisy branches without removing essential parts of the skeleton, τ_{branch} should be set between τ and 2τ. Figure 10 shows three skeleton graphs before and after the deletions. To avoid recursively pruning longer branches, we found it useful to sort the short branches in increasing order by their length, and deleting them in that order. This was especially important in the case of extremely noisy skeleton graphs such as depicted by Figures 10(a) and (b).

2.3.2 Removing Short Loops

Short loops created by thinning algorithms usually indicate isolated islands of white pixels in the template. We remove any loop from the skeleton graph if its length is below a threshold τ_{loop}.

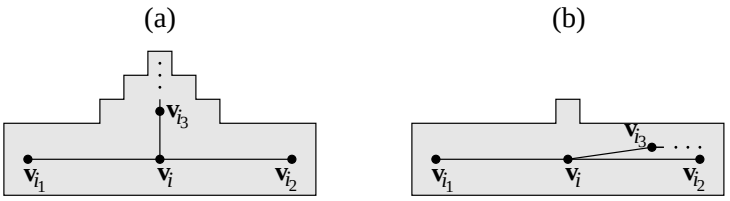


Figure 9: If the protrusion is a "real" part of the skeleton, the angles of the short branch and the connecting paths tend to be close to right angle (a), whereas if the short branch has been created by a few noisy pixels on the contour of the character, the short branch tends to slant to one of the connecting paths during the first fitting-and-smoothing step (b).

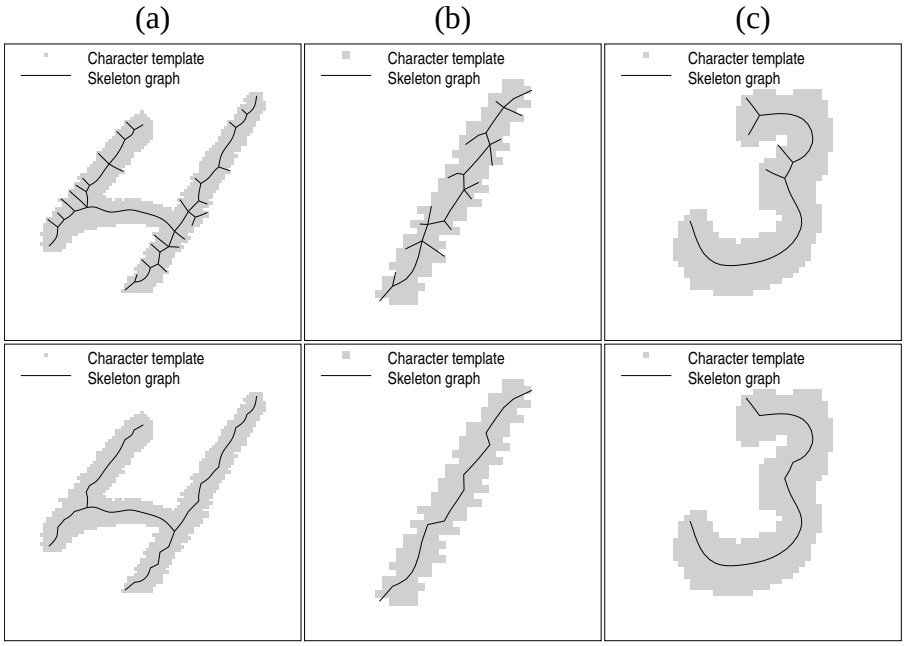


Figure 10: Deleting short branches. Skeleton graphs before (top row) and after (bottom row) the deletion.

A loop is removed by deleting the longest simple path it contains. Experiments showed that to remove most of the noisy loops without removing essential parts of the skeleton, τ_{loop} should be set between 2τ and 3τ. Figure 11 shows three skeleton graphs before and after the operation.

2.3.3 Merging Star3-Vertices

In experiments we found that if two penstrokes cross each other at a sharp angle, the thinning procedure tends to create two star3-vertices connected by a short simple path rather then a star4-vertex. To detect these situations, we analyzed the behavior of the theoretical medial axis defined by Blum and Nagel [7] at the crossing point of two ideal strokes (Figure 12). It can be determined

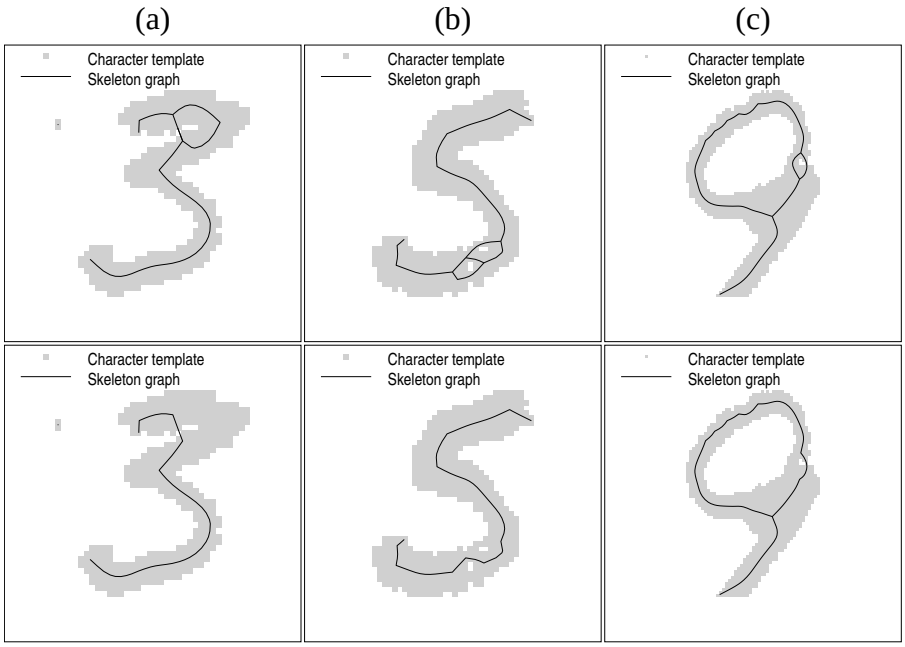


Figure 11: Removing short loops. Skeleton graphs before (top row) and after (bottom row) the removal.

analytically that the length ℓ of the short segment is

$$\ell(\alpha,\tau) = \begin{cases} \tau \frac{\frac{1}{\sin\frac{\alpha}{2}}-1}{\sin\frac{\alpha}{2}+1} & \text{if } \alpha \le 60^\circ, \\ \tau \frac{\cos\alpha}{\sin\alpha\cos\frac{\alpha}{2}} & \text{if } 60^\circ \le \alpha \le 90^\circ \end{cases}$$

where α is the angle of the strokes, and τ is their thickness.

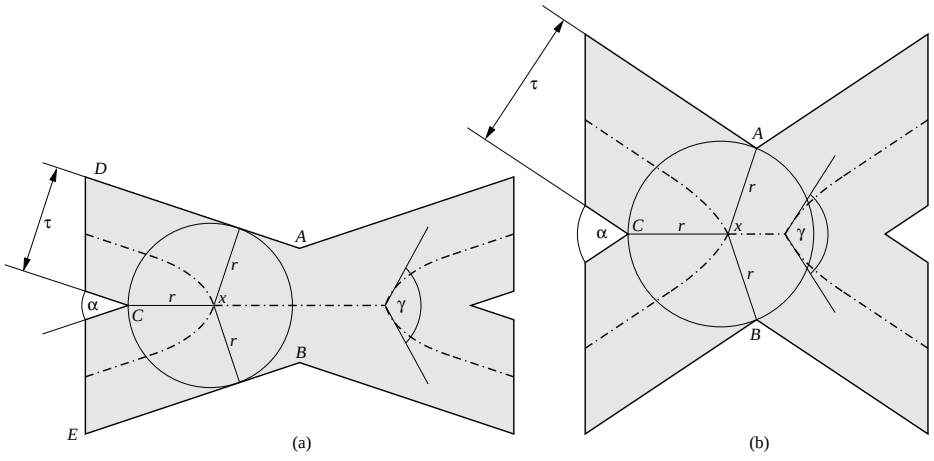


Figure 12: The behavior of the theoretical medial axis at the crossing point of two ideal strokes. Every point of the medial axis (dashed line) is the center of a maximum disc inscribed into the geometrical object. The endpoint *x* of the short path is equidistant from (a) contour lines *AD* and *BE*, and contour point *C* if $\alpha \le 60^\circ$, and (b) contour points *A*, *B*, and *C* if $60^\circ \le \alpha \le 90^\circ$.

We faced two problems when we applied the results of this analysis in practice. First, although we have a good estimation of the thickness τ, we cannot directly measure the angle α of the two strokes that originally created the image. Instead, we have an estimation for the angle γ of the joining segments of the skeleton. Interestingly, γ does not uniquely determine α (γ has a maximum of 120° at $\alpha = 60^\circ$, and decreases linearly to 90° both as α increases to 90°, and as α decreases to 0°). The second problem is that the skeleton produced by the first fitting-and-smoothing step is not ideal, so using the formulas of the ideal case to estimate α can be misguiding (an obvious problem is that γ is often larger than 120° which cannot happen according to the theoretical analysis). In practice we found that the simple formula of $\alpha = \gamma/2$ works well, so we use $\ell(\gamma/2, \tau)$ to estimate the length of the short path under the hypothesis that it was created by crossing lines. We found that most of the time the length of the short path is underestimated by $\ell(\gamma/2, \tau)$, so we scale the estimation by a factor ε_{star3} between 2 and 4.

Formally, given a path $p_{i,\ldots,j}$ with two star3-vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ as its endpoints (Figure 13), we merge $\mathbf{v}_i$ and $\mathbf{v}_j$ if $l(p_{i,\ldots,j}) < \varepsilon_{star3} \frac{\ell(\gamma_{i_1,i,i_2}/2,\tau)+\ell(\gamma_{j_1,j,j_2}/2,\tau)}{2}$. When merging $\mathbf{v}_i$ and $\mathbf{v}_j$, we first delete the path $p_{i,\ldots,j}$, and remove $\mathbf{v}_i$ and $\mathbf{v}_j$. Then we add a new vertex $\mathbf{v}_{new}$ and connect the four remaining neighbors of $\mathbf{v}_i$ and $\mathbf{v}_j$ to $\mathbf{v}_{new}$ (Figure 13). Figure 14 shows three skeleton graphs before and after the merge.

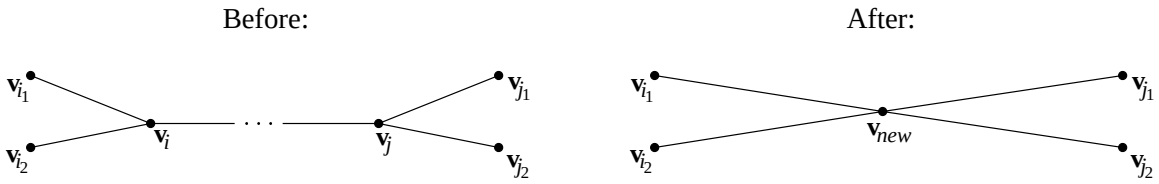


Figure 13: When merging two star3-vertices, we remove the vertices and the path connecting them. Then we add a new vertex and connect the former neighbors of the two star3-vertices to the new vertex.

2.3.4 Updating Star3 and Star4-Vertices

Initially, all the junction vertices of the skeleton are either star3 or star4-vertices. After the skeleton has been smoothed by the first fitting-and-smoothing step and cleaned by the restructuring operations described above, we update the junction vertices of the skeleton to Y, T and X-vertices depending on the local geometry of the junction vertices and their neighbors. A star4-vertex is always updated to an X-vertex. When updating a star3-vertex, we face the same situation as when degrading an X-vertex, so a star3-vertex is updated to a T-vertex if two of the angles at the vertex are between 80 and 100 degrees, otherwise it is updated to a Y-vertex. The formal rules are given in the last two rows of Table 2.

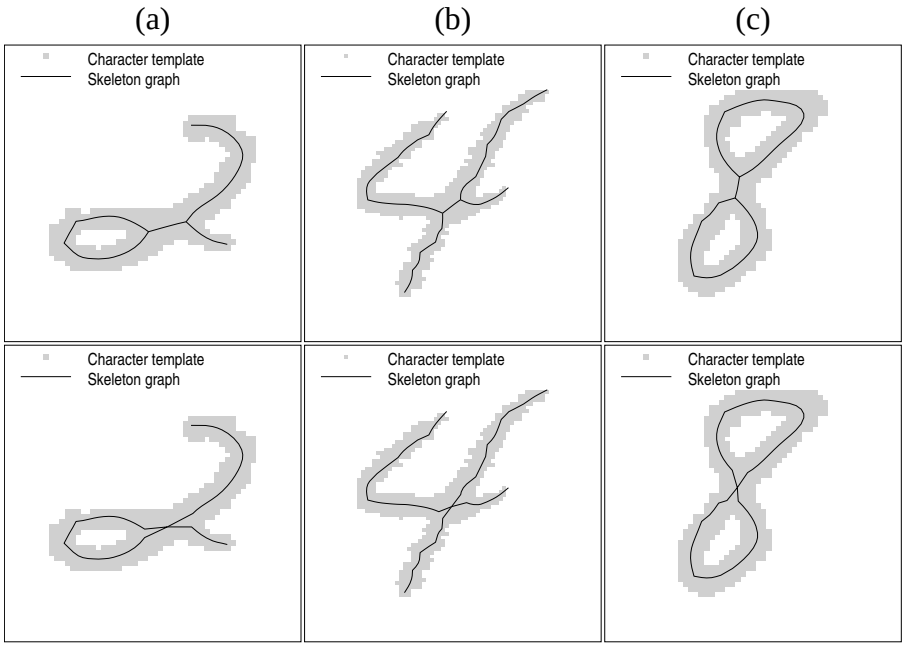


Figure 14: Merging star3-vertices. Skeleton graphs before (top row) and after (bottom row) the merge.

2.3.5 Filtering Vertices

In this step we remove line-vertices of the graph by using the polygonal approximation algorithm of Eu and Toussaint [28]. Given a path $p_{i_1,\ldots,i_\ell}$, the algorithm finds the smallest ordered set of vertices $(\mathbf{v}_{i'_1},\ldots,\mathbf{v}_{i'_{\ell'}}) \subseteq (\mathbf{v}_{i_1},\ldots,\mathbf{v}_{i_\ell})$ such that no point of the new path $p_{i'_1,\ldots,i'_{\ell'}}$ is further from $p_{i_1,\ldots,i_\ell}$ than a predetermined error tolerance τ_{filter}.

Filtering vertices is an optional operation. It can be used to speed up the fitting-and-smoothing if the character template has a high resolution since in this case the initial skeleton graph has much more vertices than it is needed for reasonably smooth approximation. It can be also used after the second fitting-and-smoothing step to improve the compression rate if the objective is to compress the image by storing the character skeleton instead of the template. In this case the filtering operation can be coupled with a smoothing operation at the other end where the character is recovered based on the skeleton graph (see Section 3.2). Figure 15 shows an example of a skeleton graph before and after the filtering operation.

2.4 Computational Complexity

The computational complexity of the algorithm is $O(n \log m)$, where n is the number of data points (black pixels), and m is the number of vertices in the skeleton graph. The computational complexity of the initialization step is proportional to the number of pixels of the image. Assuming that the number of pixels is proportional to the number of black pixels, the initialization step runs in $O(n)$

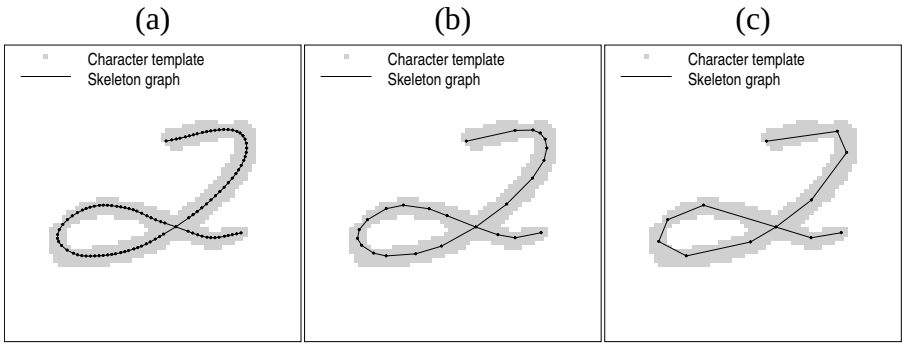


Figure 15: Filtering vertices. A skeleton graph (a) before filtering, (b) after filtering with $\tau_{filter} = 0.2$, and (b) after filtering with $\tau_{filter} = 1$.

time. The computational complexity of the restructuring step is negligible compared to the time consumed by the other three steps both in asymptotic terms ($O(m^2)$ or $O(m)$ with or without filtering vertices, respectively) and in practice.

The complexity of the fitting-and-smoothing loop (and the whole algorithm) is dominated by the complexity of the projection step. Using the naive approach described in Section 2.1.2 the creation of the Voronoi sets takes $O(nm)$ time. The step can be accelerated to $O(n \log m)$ by using the sweepline algorithm of Fortune [29] to explicitly compute the Voronoi regions, and by assigning the data points to the sets during the creation of the partition. The step can be further accelerated by exploiting the fact that most of the time the partition does not change radically between two consecutive projection steps (the algorithmic details are described in [2]).

If the sample covariances of data points belonging to each nearest neighbor set are stored, both $\Delta(\mathbf{v}_i)$ and $\nabla_{\mathbf{v}_i}\Delta(\mathbf{v}_i)$ can be computed in $O(1)$ time, so the vertex optimization step takes $O(m)$ time (as opposed to the $O(n)$ time of the naive implementation). The maintenance of these statistics can be done in the projection step when the data points are sorted into the nearest neighbor sets. The statistics must be updated only for data points that are moved from a nearest neighbor set into another in the projection step. The number of such data points tends to be very small as the algorithm progresses so the computational requirements of this operation is negligible compared to other steps of the algorithm.

If a filtering operation is applied in the restructuring step, the number of vertices of the skeleton graph is decreased, so the second fitting-and-smoothing step consumes less time than the first. The second fitting-and-smoothing also tends to iterate less since its input skeleton graph is fairly close to the optimal one. The first fitting-and-smoothing step can be accelerated by rescaling the image at a lower resolution before the initialization step. This operation decreases the number of vertices m in the initial skeleton graph, and also accelerates the initializing step. The subsequent fitting-and-smoothing steps ensure that the decreased quality of the initial skeleton does not affect the final

skeleton graph considerably (assuming that the original image is used in the fitting-and-smoothing process).

3 Experimental Results

3.1 Skeletonizing Isolated Digits

In this section we report results on isolated hand-written digits from the NIST Special Database 19 [13]. To set the parameters and to tune the algorithm, we chose 100 characters per digit randomly. Figures 16 displays five templates for each digit. These examples were chosen so that they roughly represent the 100 characters both in terms of the variety of the input data and in terms of the success rate of the algorithm. To illustrate the contrast between the pixelwise skeleton of the character and the skeleton graph produced by the principal graph algorithm, we show the initial graph (odd rows) and the final graph (even rows) for each chosen character template. The length thresholds of the restructuring operations were initialized by using the heuristics described in Section 2.3, and fine tuned by trial and error. The final values are indicated by the second row of Table 3.

Experiment	τ_{branch}	τ_{loop}	ε_{star3}	τ_{filter}
NIST digits	1.2τ	3τ	3.5	0.05
Continuous handwriting	τ	2τ	2	0

Table 3: Length thresholds of the restructuring operations in experiments with isolated digits and continuous handwriting. $\tau_{filter} = 0$ indicates that we did not filter vertices in the restructuring step.

The results indicate that the principal graph algorithm finds a smooth medial axis of the great majority of the characters. In the few cases when the skeleton graph is imperfect, we could identify two sources of errors. The first cause is that, obviously, the restructuring operations do not work perfectly for all the characters. For instance, short branches can be cut (first “6”) or star3-vertices can be merged mistakenly (fifth “4”). To correct these errors, one has to include some a-priori information in the process, such as a collection of possible configurations of skeleton graphs that can occur in hand-written digits. The other source of errors is that at this phase, we do not have restructuring operations that *add* components to the skeleton graph. For instance, the skeleton graph of the third “2” could be improved by connecting broken lines based on the closeness of their endpoints. One could also add short paths to create branches or loops that were missing from the initial graph (first “2”). This operation could be based on local thickness measurements along the graph that could point out protrusions caused by overlapping strokes in the character. The exact definitions and implementations of these operations are subjects of future research.

Figure 16: Initial (odd rows) and final (even rows) skeleton graphs of isolated numerals.

3.2 Skeletonizing and Compressing Continuous Handwriting

In this section we present results of experiments with images of continuous handwriting. We used the principal graph algorithm to skeletonize short pangrams (sentences that contain all the letters of the alphabet) written by different individuals. The emphasis in these experiments was on using the skeleton graph for representing hand-written text efficiently.

The first row of Table 4 shows the images of two pangrams written by two individuals. For the sake of easy referencing, hereafter we shall call them Alice and Bob. After scanning the images, the principal graph algorithm was used to produce the skeleton graphs shown in the second row of Table 4. Since the images were much cleaner than the images of isolated digits used in the previous section, τ_{branch} and τ_{loop} were set slightly lower than in the previous experiments. We also found that the incorrect merge of two star3-vertices has a much worse visual effect than not merging two star3-vertices when they should be merged, so we set ε_{star3} to roughly half of the value that was used in the experiments with isolated digits. The length thresholds of the restructuring operations

were set to the values indicated by the third row of Table 3. The thickness of each curve in Table 4 was set to the estimated thickness τ of the template.

To demonstrate the efficiency of representing the texts by their skeleton graphs, we applied the vertex filtering operation *after* the skeleton graphs were produced. For achieving high compression rate, τ_{filter} should be set to a relatively large value to remove most of the line-vertices from the skeleton graph. Since filtering with a large threshold has an unfortunate visual effect of producing sharp-angled polygonal curves (see Figure 15(c)), we fit cubic splines through the vertices of each path of the skeleton graph. Rows 3-5 of Table 4 show some of the results.

To be able to compute the number of bytes needed for storing the images, in the compression routine we also set the number of bits n_b used to store each coordinate of a vertex. The vertices are stored consecutively with one bit sequence of length n_b marking the end of a path. So, for example, when n_b is set to 8, the vertices of the skeleton graph are rounded to the points of a 255×255 rectangular grid, and the remaining byte is used to mark the end of a path. By using this scheme, the skeleton graph can be stored by using $N = \lceil (n_p + 2m)n_b/8 \rceil$ bytes where n_p is the number of paths and m is the number of vertices. Rows 3-5 of Table 4 show the skeleton graphs and the number of bytes needed to store the images. The numbers of paths in Alice's and Bob's texts are 143 and 74, respectively.

As a comparison, the last four rows show the compression rates achieved by several popular image compression algorithms applied on the original image. The results indicate that the images can be represented efficiently by their skeleton graphs while preserving the main features of the handwriting. For instance, if the filter threshold is set to 6τ, and 8 bits are used to store the coordinates of the vertices, the algorithm produces a skeleton that approximates the original text well while compressing the image to almost half of the size of the image compressed by using CCITT Group 4 compression (developed specifically for binary images transmitted by fax machines). Note that the bit sequence representing the skeleton graph can be further compressed by using a more sophisticated coding scheme based on traditional compression methods. Further testing and comparison, and the development of a turn-key compression method based on the principal graph algorithm are subjects of future research.

4 Conclusion

In this paper, we described a fully automatic method to find piecewise linear skeletons of handwritten characters. The main step of the algorithm is built on the polygonal line algorithm [1, 2] which approximates principal curves of data sets by piecewise linear curves. The algorithm also contains two operations specific to the task of skeletonization, an initialization method to capture

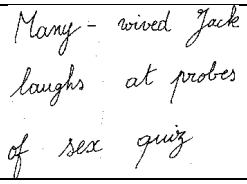
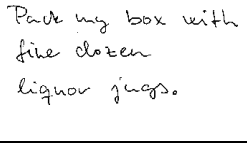
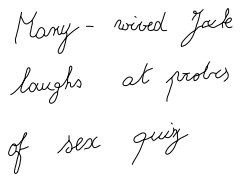
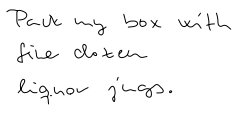
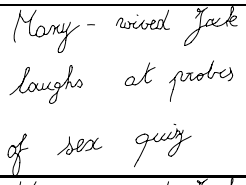
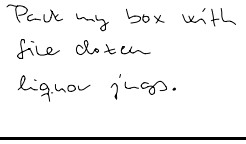
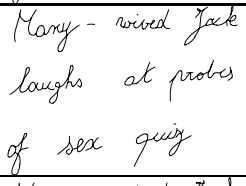
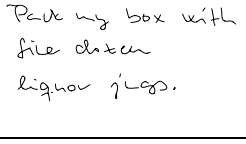
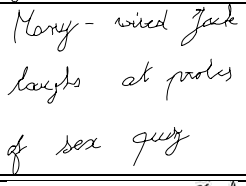
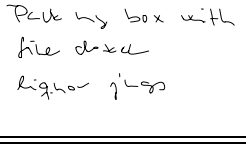
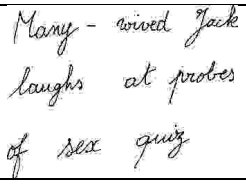
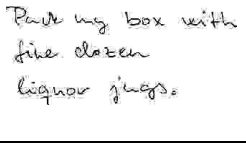
	Alice		Bob	
Original image				
Skeleton graph				
$\tau_{filter} = 1$ $n_b = 8$		$m = 436$ $N = 1015$		$m = 256$ $N = 586$
$\tau_{filter} = 2$ $n_b = 8$		$m = 372$ $N = 887$		$m = 214$ $N = 502$
$\tau_{filter} = 4$ $n_b = 6$		$m = 339$ $N = 616$		$m = 184$ $N = 332$
JPEG		$N = 3472$		$N = 2125$
GIF	lossless	$N = 2589$	lossless	$N = 1445$
Lempel-Ziv	lossless	$N = 2322$	lossless	$N = 1184$
CCITT Group 4 (fax)	lossless	$N = 1632$	lossless	$N = 992$

Table 4: Test results and compression rates with continuous handwriting. n_b is the number of bits used to store each coordinate of a vertex, m is the number of vertices, and N is the total number of bytes needed to store the image.

the approximate topology of the character, and a collection of restructuring operations to improve the structural quality of the skeleton produced by the initialization method. Test results indicate that the proposed algorithm can be used to substantially improve the pixelwise skeleton obtained by traditional thinning methods. Results on continuous handwriting demonstrate that the skeleton produced by the algorithm can also be used for representing hand-written text efficiently and with a high accuracy.

In Section 1 we described several other methods [17, 18, 15] that approach skeletonization as a graph-fitting process. Although, similarly to the principal graph algorithm, these methods also use a piecewise linear approximation of the skeleton of the character, their models substantially differ from our approach in that smoothness of the skeleton is not a primary issue in these works. Although the SOM algorithm (used by [18, 15]) and the fuzzy ISODATA algorithm (used by [17]) implicitly ensure smoothness of the skeleton to a certain extent, they lack a clear and intuitive formulation of the two competing criteria, smoothness of the skeleton and closeness of the fit, which is explicitly given by the objective function (1) in our method. In this sense our algorithm complements these methods rather then competes with them. For example, the method of [15] could be used as an alternative initialization step for the principal graph algorithm if the input is too noisy for our thinning-based initialization step. One could also use the restructuring operations described in [18] combined with the fitting-and-smoothing step of the principal graph algorithm in a "bottom-up" approach of building the skeleton graph. The possible combinations of the principal graph algorithm and these methods are issues for future work.

Another interesting line of research would be the extension of the algorithm to images of different nature, such as engineering drawings or natural images containing curvilinear features. The optimization engine of the core fitting-and-smoothing step assumes very little a-priori knowledge about the input data. The different vertex types in this step are tailored to the task of finding a smooth skeleton of a character template. In a different application, other vertex types can be introduced along the same lines. On the other hand, the initialization and restructuring steps are heavily input dependent. In a different application, these steps should be redesigned according to the nature of the input data.

Acknowledgements

We would like to acknowledge the anonymous reviewers for their helpful comments and suggestions which have improved the quality of this paper. This research was supported in part by NSERC and FCAR. The second author was also supported by the Alexander von Humboldt Foundation of Germany.

References

[1] B. Kégl, A. Krzyżak, T. Linder, and K. Zeger, "Learning and design of principal curves," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 22, no. 3, pp. 281–297, 2000.

[2] B. Kégl, *Principal Curves: Learning, Design, and Applications*. PhD thesis, Concordia University, Montréal, Canada, 1999.

[3] G. P. Dinnen, "Programming pattern recognition," in *Proceedings of the Western Joint Computer Conference*, (New York), pp. 94–100, 1955.

[4] R. A. Kirsh, L. Cahn, C. Ray, and G. J. Urban, "Experiment in processing pictorial information with a digital computer," in *Proceedings of the Eastern Joint Computer Conference*, (New York), pp. 221–229, 1957.

[5] E. S. Deutsch, "Preprocessing for character recognition," in *Proceedings of the IEE NPL Conference on Pattern Recognition*, pp. 179–190, 1968.

[6] T. M. Alcorn and C. W. Hoggar, "Preprocessing of data for character recognition," *Marconi Review*, vol. 32, pp. 61–81, 1969.

[7] H. Blum and R. Nagel, "Shape description using weighted symmetric axis features," *Pattern Recognition*, vol. 10, pp. 167–180, 1978.

[8] T. Pavlidis, "A thinning algorithm for discrete binary images," *Computer Graphics and Image Processing*, vol. 13, no. 2, pp. 142–157, 1980.

[9] N. J. Naccache and R. Shingal, "SPTA: A proposed algorithm for thinning binary patterns," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 14, no. 3, 1984.

[10] S. Suzuki and K. Abe, "Sequential thinning of binary pictures using distance transformation," in *Proceedings of the 8th International Conference on Pattern Recognition*, pp. 289–292, 1986.

[11] T. Hastie, *Principal curves and surfaces*. PhD thesis, Stanford University, 1984.

[12] T. Hastie and W. Stuetzle, "Principal curves," *Journal of the American Statistical Association*, vol. 84, pp. 502–516, 1989.

[13] P. Grother, *NIST Special Database 19*. National Institute of Standards and Technology, Advanced Systems Division, 1995.

[14] J. D. Banfield and A. E. Raftery, "Ice floe identification in satellite images using mathematical morphology and clustering about principal curves," *Journal of the American Statistical Association*, vol. 87, pp. 7–16, 1992.

[15] R. Singh, V. Cherkassky, and N. P. Papanikolopoulos, "Self-organizing maps for the skeletonization of sparse shapes," *IEEE Transactions on Neural Networks*, vol. 11, no. 1, pp. 241–248, 2000.

[16] T. Kohonen, *The Self-Organizing Map*. Springer-Verlag, 2nd ed., 1997.

[17] S. Mahmoud, I. Abuhaiba, and R. Green, "Skeletonization of arabic characters using clustering based skeletonization algorithm (CBSA)," *Pattern Recognition*, vol. 24, no. 5, pp. 453–464, 1991.

[18] A. Datta and S. K. Parui, "Skeletons from dot patterns: A neural network approach," *Pattern Recognition Letters*, vol. 18, pp. 335–342, 1997.

[19] J. Bezdek and J. Dunn, "Optimal fuzzy partitions: A heuristic for estimating the parameters in a mixture of normal distributions," *IEEE Transactions on Computers*, vol. 24, no. 4, pp. 835–838, 1975.

[20] M. P. Deseilligny, G. Stamon, and C. Y. Suen, "Veinerization: a new shape description for flexible skeletonization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 5, pp. 505–521, 1998.

[21] S.-C. Zhu, "Stochastic jump-diffusion process for computing medial axes in Markov random fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 11, pp. 1158–1169, 1999.

[22] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, vol. COM-28, no. 1, pp. 84–95, 1980.

[23] S. P. Luttrell, "Derivation of a class of training algorithms," *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 229–232, 1990.

[24] S. T. Roweis, "EM algorithms for PCA and SPCA," in *Advances in Neural Information Processing Systems* (M. I. Jordan, M. J. Kearns, and S. A. Solla, eds.), vol. 10, The MIT Press, 1998.

[25] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society, Series B*, vol. 61, no. 3, pp. 611–622, 1999.

[26] A. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society Series B*, vol. 39, pp. 1–38, 1977.

[27] S. W. Lee, L. Lam, and C. Suen, "A systematic evaluation of skeletonization algorithms," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 5, pp. 1203–1225, 1993.

[28] D. Eu and G. T. Toussaint, "On approximating polygonal curves in two and three dimensions," *CVGIP: Graphical Models and Image Processing*, vol. 56, pp. 231–246, May 1994.

[29] S. Fortune, "A sweepline algorithm for Voronoi diagrams," *Algorithmica*, vol. 2, pp. 153–174, 1987.