2016-05-01

Intelligent Pen: A least cost search approach to Stroke Extraction in Historical Documents

Kevin L. Bauer
Brigham Young University - Provo

Intelligent Pen: A Least Cost Search Approach to Stroke Extraction in Historical Documents

Kevin L. Bauer

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

William A. Barrett, Chair
Ryan Farrell
Tony Martinez

Department of Computer Science

Brigham Young University

May 2016

ABSTRACT

Intelligent Pen: A Least Cost Search Approach to Stroke Extraction in Historical Documents

Kevin L. Bauer
Department of Computer Science, BYU
Master of Science

Extracting strokes from handwriting in historical documents provides high-level features for the challenging problem of handwriting recognition. Such handwriting often contains noise, faint or incomplete strokes, strokes with gaps, overlapping ascenders and descenders and competing lines when embedded in a table or form, making it unsuitable for local line following algorithms or associated binarization schemes. We introduce Intelligent Pen for piece-wise optimal stroke extraction. Extracted strokes are stitched together to provide a complete trace of the handwriting. Intelligent Pen formulates stroke extraction as a set of piece-wise optimal paths, extracted and assembled in cost order. As such, Intelligent Pen is robust to noise, gaps, faint handwriting and even competing lines and strokes. Intelligent Pen traces compare closely with the shape as well as the order in which the handwriting was written. A quantitative comparison with an ICDAR handwritten stroke data set shows Intelligent Pen traces to be within 0.78 pixels (mean difference) of the manually created strokes.

Keywords: Handwriting, Stroke Extraction, Document Processing

ACKNOWLEDGMENTS

Special thanks to FamilySearch for providing a valuable dataset of handwritten signatures.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

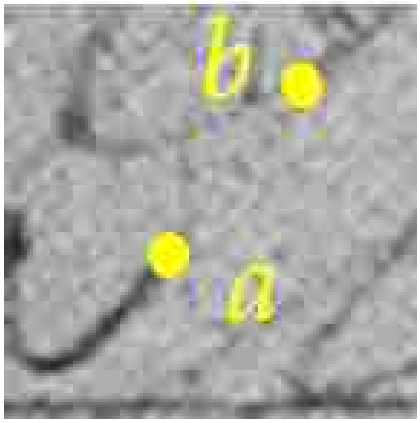
A subject of increasing interest in document processing and analysis is the movement to index the handwriting contained in scanned images of documents. Many of these documents are historical in nature such as census records, birth and death records, parish and church records, journals, marriage certificates, and lists of passengers on ships. The advent of the digital age has led to a revolution in the handling of these records. Millions of documents that could previously only be obtained by visiting and searching through libraries, churches, and government offices are now being digitized and made available via the Internet.

A key player in this process is FamilySearch, the world's largest genealogical organization. As part of a worldwide effort to help people discover and connect to their ancestors, FamilySearch is involved in a massive effort to digitize and transcribe their entire collection of historical and genealogical documents, which is contained on more than 2.5 million rolls of microfilm. This transcription process, known as indexing, allows users to digitally search through millions of documents online. In 2015 260,000 volunteer indexers[11] were involved in the manual transcription of these records, using human expertise to identify and transcribe millions of names, dates, places, etc. However, even these efforts are insufficient. New records are currently being discovered and scanned at about three times the rate that volunteers can index them. The difference is even greater when considering non-English-language documents.[12] Familysearch is not alone in this problem. Other genealogical organizations such as Ancestry.com are also involved in indexing initiatives.

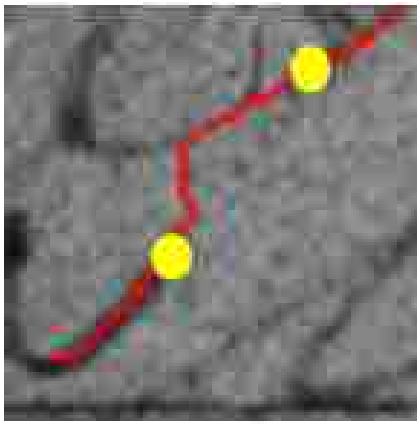
Non-genealogical groups are also being forced to confront the problem. In the face of the mounting backlog of digitized documents becoming available, historical document processing has become a key problem for government agencies, librarians, archivists, historians, and genealogists around the world. For example, the European Union is currently funding the tranScriptorium Project with the goal of creating more "efficient and cost-effective solutions for the indexing, search, and full transcription of historical handwritten document images"[4]. The project includes datasets in Dutch, Spanish, German, and English ranging from 16th century birth records to handwritten books and essays to early 20th century court documents. The breadth of topics and history covered by the project make it clear that research in this field can have far-reaching benefits in a variety of domains.

Prior to tranScriptorium the EU also allocated 16.5 Million Euros to fund the IMPACT Project, the goal of which was to improve access to historical texts by digitizing Europe's many culturally important documents. In addressing the scale of the problem, they noted that "Today, a number of advanced libraries in Europe are scanning millions of pages each year.... However, these efforts can tackle only a fraction of the total heritage available in cultural memory organisations.[2]" In the business world, Google is addressing the issue of document scanning and indexing, most notably in connection with the Tesseract OCR engine [3].

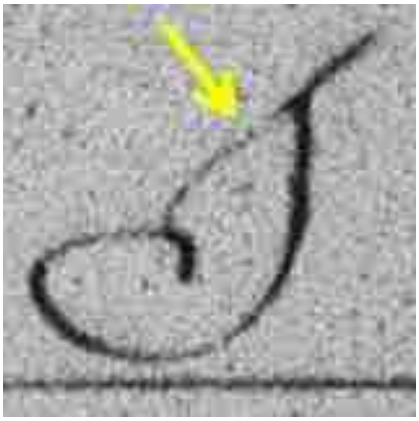
While the above projects have led to many advancements in historical document processing, as of this writing no reliable end-to-end system exists for converting a historical document image into a fully indexed record. When such systems have been proposed in the past they have typically included a number of steps, each of which poses a unique challenge.[8] Among these challenges is that of properly segmenting the image into words and extracting the relevant stroke information from the handwriting. By their nature, historical documents pose many challenges for stroke extraction, including faint strokes, strokes with gaps, form lines, and ascenders and descenders (Figures 1.1 and 1.2).



(a) A stroke with a gap between points a and b. (143 x 145 pixels)



(b) Bridging the gap.



(c) A faint stroke with low SNR. (159 x 159 pixels)

(d) Intelligent Pen follows the faint stroke.

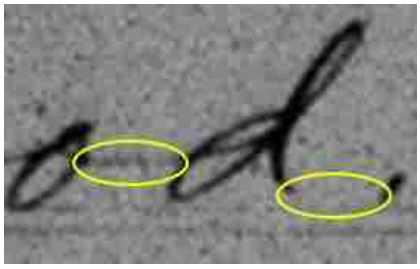
Figure 1.1: Intelligent Pen overcomes many of the pitfalls common to historical document images.

While many algorithms for stroke extraction exist, their effectiveness is limited to newer documents that are cleanly scanned with little degradation or that are born digital. However, the same problems that make it difficult to perform stroke extration on historical documents are also what make performing handwriting recognition difficult in this domain. By extracting the strokes from the document in a way that is robust to noise, low-contrast, gaps, and form lines we hope to improve the accuracy of handwriting recognition as well.

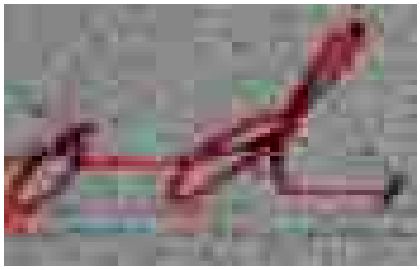
At its core, any system for processing historical documents would need a reliable method for performing handwriting recognition across a variety of document types and

(a) Handwriting with Form Lines and false lines causing interference. (150 x 95 pixels)

(b) Intelligent Pen ignores the form lines, extracting only the handwriting.



(c) Faint strokes and gap with low SNR. (266 x 167 pixels)



(d) Intelligent Pen recovers faint or missing strokes.

Figure 1.2: More problems common to historical document images.

languages. As a result, (semi)automated indexing/transcription and word spotting are still an area of active research, particularly with historical documents.

Many of these techniques rely on extracted features, profiling of the handwriting, or other abstractions of the handwriting. Very few have addressed the difficult problem of directly tracing the handwriting, which gives us direct and primary access to the signal itself. This thesis addresses this challenging problem.

Chapter 2

Related Work

Typically the goal of stroke extraction algorithms is to be able to apply on-line handwriting recognition methods to static, offline handwriting images. This is usually done by using a medial axis transform or some other thinning algorithm to extract a skeleton: a pixel-wide representation that outlines the basic shape of the handwriting. Figure 2.1 gives an example of a skeleton obtained by line thinning. Various techniques exist for extracting word skeletons, such as diffusion maps [9] or line thinning [13, 16].

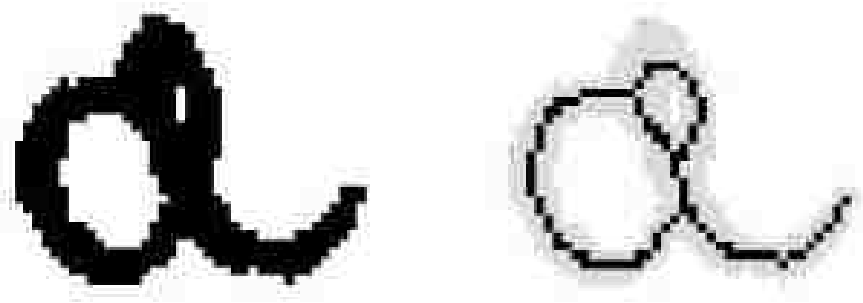


Figure 2.1: An example of a handwritten lower-case 'a' and its skeleton, obtained by line thinning. (Taken from L'Homer[16]).

Once the skeleton is obtained, some tracing method is used to find an ordered path through the skeleton. As noted by Qiao and Yasuhara [21], skeleton tracing methods fall into two main categories: local tracing methods, which choose a direction at each junction in the skeleton based on local heuristics, and global searches, which create a graph model of the skeleton and then use search techniques to find an optimal path through the text.

The bulk of the work examined falls into these two categories, with a few exceptions. Yan and Leedham [26] employ an adaptive thresholding technique that subdivides the image into small regions and then examines features such as the stroke angle and edge strength to set the thresholding value for that region. Although their method works well for separating handwriting pixels from the background it does not account for the presence of form lines and does not obtain the actual strokes of the handwriting.

Clawson and Barrett [7] attempt to distinguish handwriting pixels from form pixels by first registering a group of similar documents to a template image having the same form lines but with the handwriting removed. A sliding window is then compared against both the template and the source image, and areas where the document differs from the template are marked as handwriting. However this approach gives incorrect results for documents with folds, tears, or other non-handwriting areas that differ from the template.

2.1 Skeletonization and Local Search

Daher et al [9] obtain the skeleton using a diffusion map. They then perform an ordered trace of the skeleton by first finding a start point, then using the gradient to calculate a direction and step distance. However, their goal is only to extract graphemes, not complete strokes, and their results are given in terms of the number of graphemes extracted per page, which makes it unclear how effective their method is in obtaining complete strokes.

Pavlidis [19] uses a line adjacency graph, scanning horizontally on a binary image and finding sections of three or more adjacent dark pixels. Each of these sections becomes a node in the line adjacency graph and is connected to adjacent nodes immediately above and below it. Local heuristics are then used to identify path and junction sections to create a simplified graph. Although faster than other line thinning techniques, this method relies on binarization and performs poorly on diagonals and curves. Their examples were also primarily machine printed characters, so it's unclear how suitable their method is for cursive handwriting.

Lee and Pan [15] extract the stroke order by searching for an endpoint, then using local heuristics to trace back to the start point. They then reverse the order of this back-traced path and merge with adjacent paths to find a global ordering of the pixels.

In [6], Boccignone et al. use a line-thinning algorithm that preserves the local thickness of the line. This is done by labeling each pixel of the skeleton with its 8-distance from the nearest background pixel.

One unique variation is proposed by Lallican and Viard-Gaudin [14]. They use the grayscale image to extract edge pixels, then match each edge pixel with the one opposite it. The midpoints of the cross-sections of these pixels are used to form a pseudo-skeleton. They then extract full strokes by applying a Kalman filter to the pseudo-skeleton. Finally, they use a smoothness function to divide or combine these curves. While their method is successful in following strokes that undergo several changes in curvature it does not account for variations in pixel intensity. Their tests are confined to clean, high-contrast images where the intensity of the handwriting is fairly dark and homogeneous, making it unclear how they would perform on historical documents.

Doerman and Rosenfeld [10] also employ a method that uses cross-section lines connecting corresponding edge points. In their case, the handwriting is not represented as a skeleton but as a series of cross-sections. This allows easy computation of the direction of the handwriting (which should be perpendicular to the cross-section) as well as the thickness (which is the length of the cross-section). However, their method relies heavily on local heuristics, making it brittle in the presence of noise and degradation such as are found in historical documents.

2.2 Graph Search and Optimal Path Finding

Skeletons are often represented as graphs where each node is a path, junction, or endpoint section of the handwriting. These skeleton extraction techniques are then combined with

graph search methods, most of which rely on the gradient and local stroke direction to determine which path to follow at each branch node.

Tan et al. [23] apply the skeletonization and graph search approach to handwritten Chinese characters with the purpose of extracting the common components that make up each character. They test their method with four different skeleton extraction methods. The top performer was the wavelet transform, which, combined with their method, allowed them to accurately extract 90.7% of the strokes in their dataset of 341 Chinese characters, with a character-level accuracy of 97%. Their dataset consisted of clearly defined characters with thick strokes and no degradation issues.

Viard-Gaudin et al. [24] used a windowed scan to break the handwriting into segments, each of which is small enough to be represented with a feature vector. They then used a graph search to connect the segments into strokes.

Qiao and Yasuhara propose multiple methods in [20], [22], and [21]. In [20] they use a combination of local heuristics and a global smoothness algorithm to compute the optimal path through the skeleton. In [22] they use a bi-directional search, picking a start and end point, then searching forward from the start point and backward from the end point and finally merging the two paths.

[21], [13], and [5] all treat the path traversal as an optimal Euler path problem. An Euler path is defined as a path through an undirected graph that passes through each node exactly once. In each case the authors were concerned primarily with stroke ordering and their methods assume that the image can be cleanly binarized without loss of data. For example, [21] uses a set of images of handwriting they wrote and scanned themselves as well as static images converted from a database of digitally obtained, online data. Similarly [13] uses clean images, and [5] only provides examples of the graphs obtained from the handwriting, not the handwriting itself.

2.3 Image Segmentation

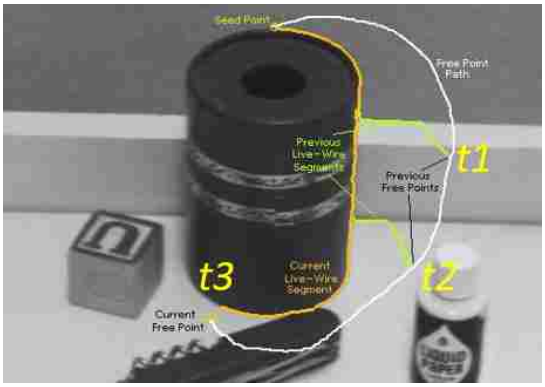
The key assumption made by the skeletonization algorithms discussed above is that the skeleton can be successfully extracted without loss of data or the introduction of noise, and the publications contain sparse information on what validation has been performed to this effect. To overcome these challenges a new approach is needed that is more robust to noise and works with older, degraded handwriting, and is tolerant to the existence of form lines that overlap the handwriting. Rather than create a skeleton or graph representation of the image, we propose an algorithm that would operate directly on the grayscale pixels of the original image.

The field of image segmentation provides inspiration for such an approach. In [17] Mortensen and Barrett introduce Intelligent Scissors, which uses a least-cost search algorithm to find the edges of an object of interest in an image based on a few seed points. This is done using a weighted sum of several image features.

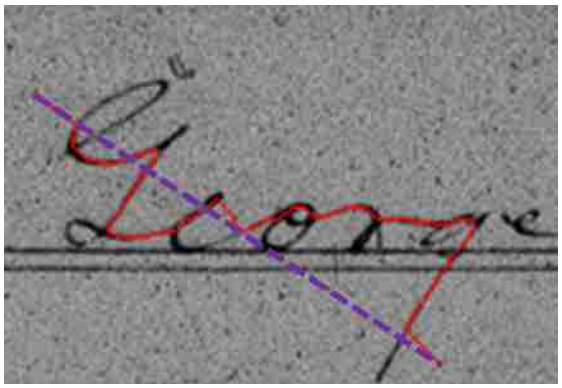
Intelligent Scissors is a user-guided system, requiring the user to guide the cursor around the object of interest. As the user guides the cursor around an object in the image, Intelligent Scissors creates a path that snaps to the boundary of the object, as shown in Figure 2.2a

By choosing different features for the path-finding algorithm we repurpose the idea behind Intelligent Scissors to create a new algorithm for extracting strokes of handwriting, one that is piecewise optimal, operates on the original grayscale image, is robust to noise, degradation, and other features of historical documents, and does not require user interaction. Because it takes its inspiration from Intelligent Scissors, we adopt the name Intelligent Pen for our approach. Figure 2.2b gives an example of how Intelligent Pen snaps to handwriting in the same way that Intelligent Scissors snaps to object boundaries.

One important feature of Intelligent Scissors is the idea of path cooling. Note how from t1 to t2 to t3 the first section of the path (orange) does not change. Intelligent Scissors takes advantage of this by freezing a section of the path if it does not change after a certain



(a) Intelligent Scissors is a user-guided system. It uses a minimum-cost path search to snap the path to the object's boundary.



(b) Intelligent Pen also uses a minimum-cost path search, only tuned to snap to handwriting strokes instead of edges. (435 x 297 pixels)

Figure 2.2

number of time steps. The end of the path then becomes a new seed point. This provides inspiration for a key element of Intelligent Pen: path consensus. Unlike Intelligent Scissors, which is a user-guided system, Intelligent Pen automatically determines seed points and stopping criteria. Likewise, it functions automatically by replacing user input and path cooling with the idea of path consensus. The next chapter examines how this is done.

Chapter 3

Intelligent Pen

Intelligent Pen performs a minimum cost path search using a cost function that can be tuned for a variety of handwriting types, including the domain of historical document processing. The goal is to create a complete trace of the handwriting for which the minimum cost paths are the ones that pass through the handwriting, ignoring noise and properly following the original path of the author's pen, even in the presence of gaps and low-contrast segments.

Section 3.1 describes the wavefront expansion process, which is core to the Intelligent Pen algorithm. Section 3.2 explains how the Distance Transform is used to enhance the image, making it easier to find the center of the handwriting strokes. Sections 3.3 and 3.4 describe the cost function, which is used to provide better discrimination between handwriting and background. Since Intelligent Pen needs to work without user interaction, seed points are chosen automatically, and this process is described in Section 3.5. Section 3.6 explains how wavefronts are expanded in parallel and the advantages this provides. Section 3.7 presents the use of path consensus and how that allows Intelligent Pen to perform autonomously. Section 3.8 describes the creation of new seed points, allowing paths to continue to propagate. Sections 3.9 and 3.10 present stopping criteria for Intelligent Pen and path pruning to eliminate false path branches or extensions. Section 3.11 discusses the use of additional passes to catch strokes that may have been missed on the first pass. Section 3.12 discusses how paths are connected and ordered, and Section 3.13 presents the user interface and options in the Intelligent Pen software.

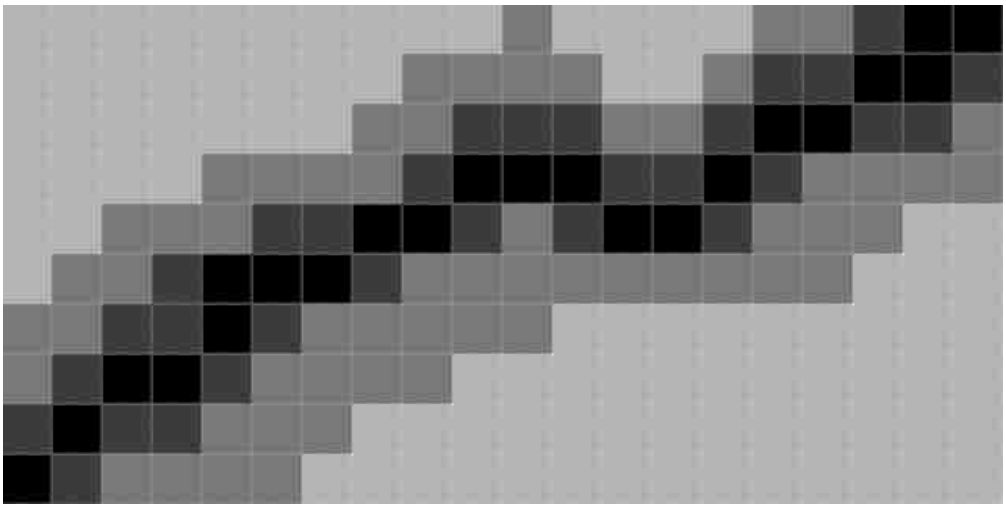
3.1 Wavefront Expansion

The minimum cost path search uses a variation of Dijkstra's depth-first search algorithm. Starting from a seed point, Intelligent Pen explores outward, giving preference to lower-cost pixels. At any given point we refer to the explored part of the image as a cost "wavefront" because it expands over the image like a wave.

Figure 3.1 and Algorithm 1 give an example of how the cost wavefront expands. We begin by defining a static cost for each pixel in the image. Cost is directly proportional to pixel intensity, with darker pixels having lower cost and lighter pixels having higher cost, as in Figures 3.1a and 3.1b. Wavefront expansion begins with an automatically determined seed point (Section 3.5) as shown in Figure 3.1b. The seed point is then expanded, and all its neighbors are added to the wavefront (purple, Figure 3.1c-3.1e). Costs on the wavefront are maintained in sorted order and the node with the lowest cost is the next to be expanded (Figure 3.1f). Any pixels neighboring this node that are not already in the graph are added, and their cost is set to be the cumulative cost of the shortest path back to the seed point (Figure 3.1g). This is done by adding the value from the initial cost matrix to the cumulative cost for the node being visited. The next lowest cost unexpanded pixel is chosen (Figure 3.1h), and so forth until a stopping criterion is reached. As the wavefront expands, it follows the darkest (lowest cost) parts of any strokes (Figure 3.1i). Each node stores a "backpointer" to the neighbor lying on its shortest path to the seed point, so at any time we can easily obtain the shortest path to the seedpoint from anywhere in the wavefront by following the backpointers (Figure 3.1j).

It should be noted that due to Bellman's principle of optimality [25] every sub-path of an optimal path is itself an optimal path between its endpoints. We exploit this principle to stitch together piece-wise optimal paths.

Since the points with the lowest cumulative cost are expanded first, each wavefront tends to do a soft fill on the handwriting as it expands in all directions (Figures 3.1i and



(a) Enlarged view of a line segment.

18	18	18	18	18	18	18	18	18	18	12	18	18	18	18	12	12	6	3	3
18	18	18	18	18	18	18	18	12	12	12	12	18	18	12	6	6	3	3	6
18	18	18	18	18	18	18	12	12	6	6	6	12	12	6	3	3	6	6	12
18	18	18	18	12	12	12	12	6	3	3	3	6	6	3	6	12	12	12	12
18	18	12	12	12	6	6	3	3	6	12	6	3	3	6	12	12	12	18	18
18	12	12	6	3	3	3	6	12	12	12	12	12	12	12	12	12	18	18	18
12	12	6	6	3	6	12	12	12	12	12	18	18	18	18	18	18	18	18	18
12	6	3	3	6	12	12	12	12	18	18	18	18	18	18	18	18	18	18	18
6	3	6	6	12	12	12	18	18	18	18	18	18	18	18	18	18	18	18	18
3	6	12	12	12	12	18	18	18	18	18	18	18	18	18	18	18	18	18	18

(b) Initial Cost Matrix: Cost $\sim$ pixel intensity. Orange seed point.

18	12	12	12	12
12	12	6	6	6
12	6	3	3	3
3	3	6	12	6
6	12	12	12	12

(c) Each neighbor sets a back-pointer.

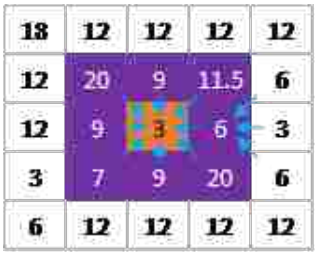
18	12	12	12	12
12	12	9	6	6
12	9	3	6	3
3	3	9	12	6
6	12	12	12	12

(d) Cumulative cost is calculated.

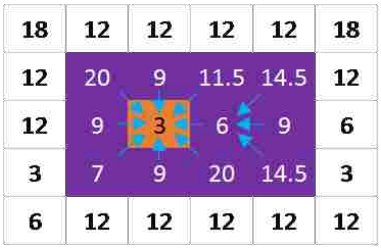
18	12	12	12	12
12	20	9	11.5	6
12	9	3	6	3
3	7	9	20	6
6	12	12	12	12

(e) Diagonal Cost given by C_n * $\sqrt{2}$.

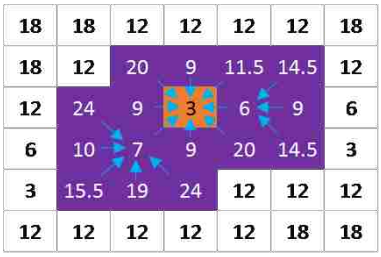
3.9). The following sections describe how paths are grown and coalesced from the collision of multiple wavefronts to obtain a complete trace of the handwriting.



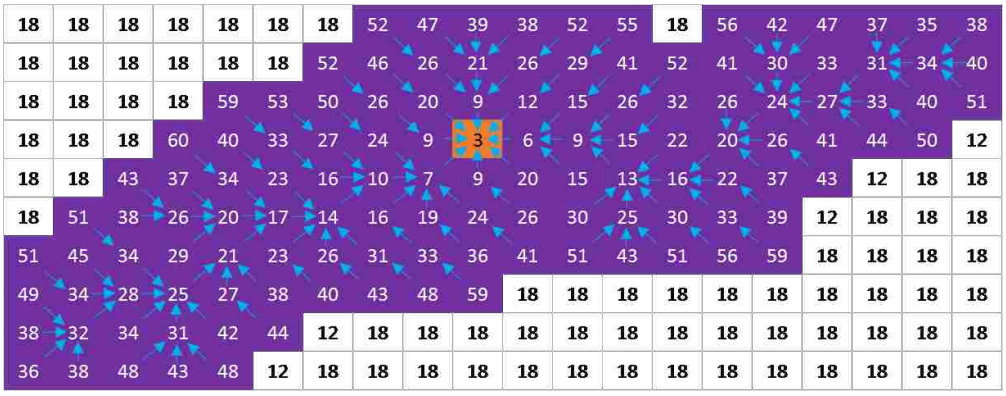
(f) Expand node with minimum cumulative cost (6).



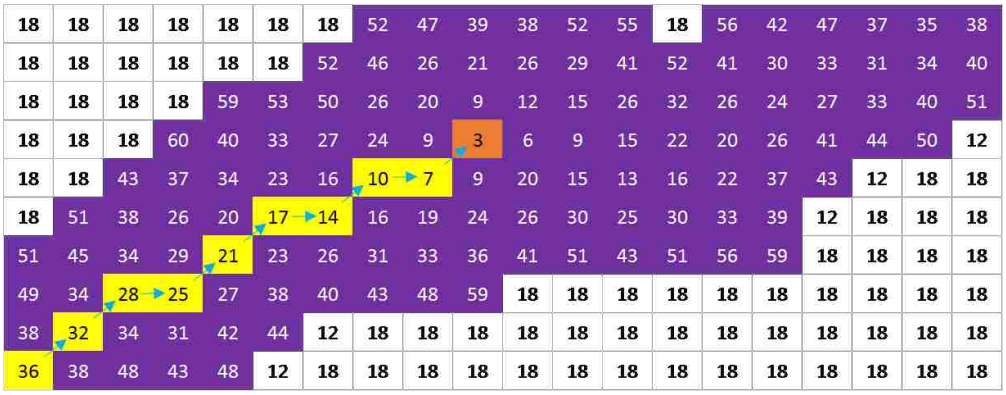
(g) Cumulative costs updated.



(h) Expand Node with next lowest cumulative cost (7, lower left).



(i) The wavefront expands preferentially along lower-cost (darker) strokes.



(j) The minimum cost path to the seed point can be found for any point in the wavefront by following its backpointers.

Figure 3.1: Wavefront Expansion Algorithm.

Algorithm 1 Wavefront Expansion Algorithm

```
1: W ← startPoints // W: set of all points in the wavefront
2: B ← startPoints // B: cost-ordered set of boundary points
3:
4: while No stopping criteria reached do
5:     // Get the lowest-cost, unvisited point
6:     // (53 In Fig 3.1c, 156 in 3.1f, and 159 in 3.1h)
7:     p ← B[0]
8:     neighbors = getNeighbors(p)
9:
10:    for n in neighbors do
11:        if n not in W then
12:            W.add(n)
13:            B.add(n)
14:            // calculate cost based on distance (Fig. 3.1e)
15:            if n.x ≠ p.x and n.y ≠ p.y then
16:                n.cost ← n.cost * √2
```

3.2 Distance Transform

Before computing the costs for each pixel we apply a distance transform DT to the image by first applying Otsu thresholding to binarize the image at threshold t, and then assigning each pixel with an intensity less than t an integer value representing its distance to the nearest white pixel. The local maxima of the DT form the center of any thresholded line. We then scale the DT by s and subtract it to from the original pixel value. For the examples in this paper $s = 2$. Because this process can produce values of less than zero we re-scale the image to keep all the pixel values between 0 and 255.

The idea behind using the distance transform is to serve as a tie-breaker for thick strokes so that the algorithm will snap to the center of the line rather than having to choose between multiple, equally low cost paths through a thick stroke. For faint strokes not visible after Otsu thresholding the distance transform has no meaning and the original pixel value is used for the cost function. Figure 3.2 shows the effect of the distance transform on the cost profile of a line. If the line is composed of pixels with uniform intensity then the stroke's profile would look something like Figure 3.2a. The DT has a trough-shaped profile (Figure

(a) Strokes with uniform pixel intensity have a flat profile.

(b) The profile of the distance transform DT is trough-shaped.

(c) Subtracting the DT from the original image snaps the local minimum to the stroke's center.

Figure 3.2: Using the Distance Transform (DT) to differentiate between equal-cost paths.

3.2b), so subtracting it from the original image gives the cost profile of each stroke a clear minimum at the stroke's center (Figure 3.2c).

3.3 Cost Function

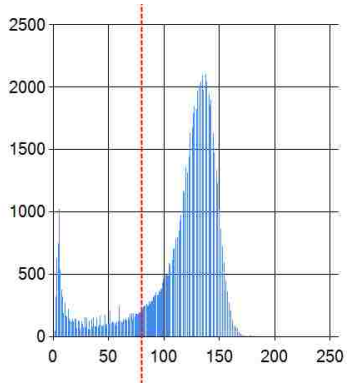
In the example shown above we assign a cost for each pixel based on how light or dark it is (intensity), which works well in contemporary documents. However, in historical documents there is often little difference in intensity between the handwriting and the background. As a result, using only the intensity to assign cost led to one background pixel being expanded for every two or three handwriting pixels. Since we want the wavefront to tightly follow the handwriting this presented a problem, suggesting that a non-linear cost function might be in order.

Initially we tried using an exponential function to strongly penalize light pixels. However, testing showed that results were overly sensitive to the degree of the exponential function, which needed to be tuned manually for each dataset. This caused us to look for a more robust and adaptive approach that could be based on the image pixels themselves, such as the image histogram.

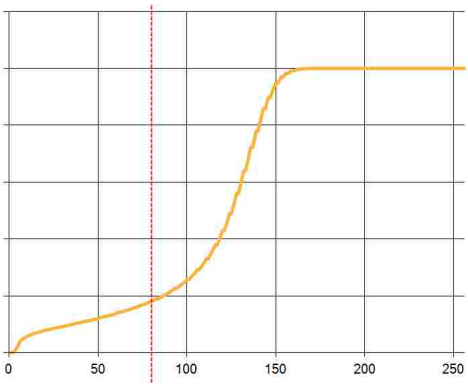
The nature of handwriting images is to have a histogram with two peaks: a large peak for the background pixels and a smaller peak for the foreground (handwriting) pixels (Figure 3.3a). Numerically integrating the histogram produces the curve shown in Figure 3.3b, with a roughly exponential increase in cost for values above the Otsu threshold ([18]). While this worked somewhat well, it had the problem of giving nearly black pixels a cost not much lower than pixels at the Otsu threshold. But we needed the cost of the background pixels to be 10-20 times greater than the darker ink pixels. This suggested a sigmoidal cost function, as shown in Figure 3.3c, with the advantage that it could be roughly modeled on the histogram while also providing better performance in practice.

The cost C_{xy} for each pixel p in the image using the sigmoidal cost function is defined in Equation 3.1

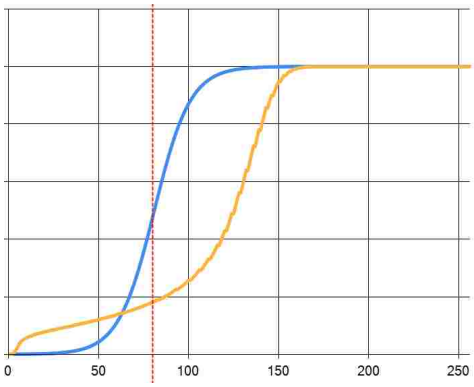
$$C_{xy} = \frac{1}{(1 + e^{-k(i-t)})} + (0.75 * F_{xy} + 0.0001) \tag{3.1}$$



(a) The histogram of a handwriting image is typically a bimodal Gaussian. The dashed red line marks the Otsu threshold t.



(b) Integrating the Histogram gives a roughly sigmoidal curve, increasing exponentially near the Otsu threshold.



(c) To model this, the cost is calculated using a sigmoidal function (blue) centered at the Otsu threshold t.

Figure 3.3: The histogram provides inspiration for the cost function.

where i is the grayscale intensity of pixel p after subtracting the DT and rescaling (Section 3.2) and t is the Otsu threshold of the image (as defined in [18]). Subtracting t from i in the equation creates the sigmoidal cost function centered at t shown in Figure 3.3. The parameter k determines the steepness of the curve, with lower k giving a gentler curve and higher k giving a steeper curve. After some experimentation a value of $k = 0.1$ was shown to work well and was used for the examples in this thesis. F_{xy} is a cost applied for form lines and varies for each row depending on the distance to the nearest form line (Equation 3.3) Finally, we add a constant value of 0.0001 to the cost so that, in the presence of pure black (zero cost) paths we give preference to shorter paths. .

When expanding a pixel p in the wavefront, the cumulative cost C_c for each neighbor n of p is calculated as follows:

$$C_c = d_n * C_n + C_p \tag{3.2}$$

The distance cost d_n is 1 for four-connected neighbors of p and $\sqrt{2}$ for diagonal neighbors of p.

Before running the algorithm we manually specify the y coordinates of the centers of any form lines and calculate F_{xy} for all pixels within 5 rows according to the following equation:

$$F_{xy} = \begin{cases} \frac{5 - |(y - y_f)|}{5} & \text{if } |(y - y_f)| < 5 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

In Equation 3.3 y_f is the center point of the nearest form line. As shown in 3.1 F_{xy} is multiplied by 0.75. This value was chosen after experiments showed that it works well at allowing Intelligent Pen to find strokes that cross the form lines while discouraging it from following the form lines themselves.

The cost function can also be described in terms of its effect on the expanding cost wavefront, (described in Section 3.1). Ideally, we want the wavefronts to expand in a soft form fill of the handwriting. The sigmoidal cost function fits this nicely. In the limit as k grows larger the cost function approaches a simple Otsu thresholding. Conversely, lower values of k differentiate less and less between foreground and background pixels. We therefore chose a k that would allow the algorithm to explore approximately 1 background point for every 10 foreground points. Section 3.1 explains how this is done.

3.4 Creating an Image-Specific Cost Function Using Interactive Training

At this point it's worth noting that, while the Sigmoidal cost function described in Section 3.3 works well on a variety of image types, good results can also be obtained by using an exponential cost function, then manually tuning it to fit the desired image type. When doing this, Equation 3.1 is modified so that the cost C_{xy} for each pixel is given by

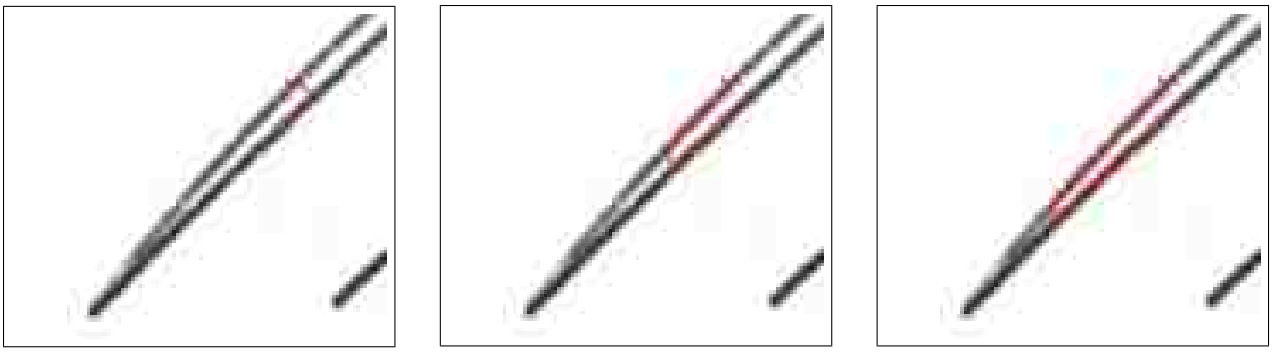
$$C_{xy} = i^b + (0.75 * F_{xy} + 0.0001) \tag{3.4}$$

where b is a manually defined exponent. If b is too small then the algorithm will tend to be overzealous in bridging gaps because the penalty for light pixels isn't high enough. However, if b is too large then Intelligent Pen will fail to jump across any gaps, even when it should. Figures 3.4 - 3.6 show some examples of the shortest path between two points before and after adjusting b.

(a) With a value of $b = 2.51$ Intelligent Pen will incorrectly jump across the gap.

(b) Adjusting b to 2.52 gives the lowest possible value for b that won't incorrectly jump across the gap.

Figure 3.4: Tuning b to avoid jumping the gap.

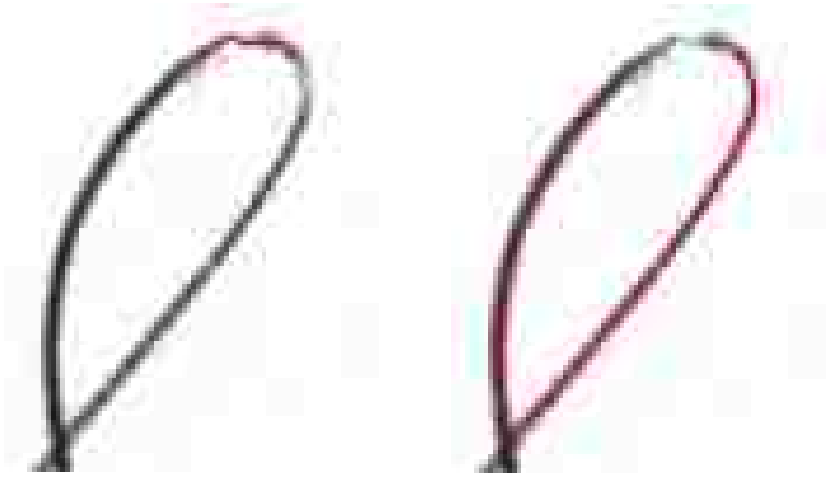


(a) $b = 3.37$ fails to follow the handwriting when strokes are close.

(b) Adjusting b to 3.38 gets more of the stroke.

(c) Beyond b = 3.94 no further improvements can be made.

Figure 3.5: Different values of b jump more or less of the gap.



(a) With a value of $b = 5.08$ Intelligent Pen will correctly jump across the gap.

(b) Increasing b to 5.09 causes it to take the long way around, avoiding a valid stroke because the cost is too high.

Figure 3.6: Increasing b too much causes poor performance on gaps and light strokes.

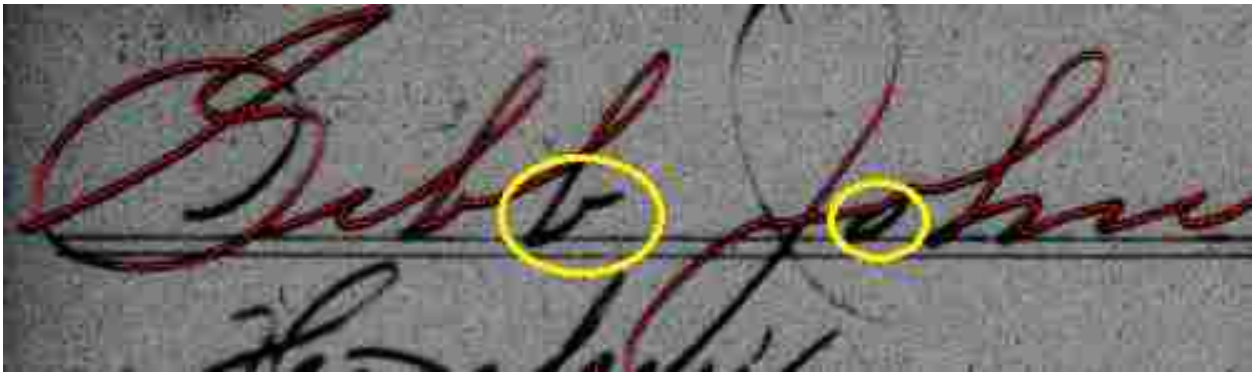
After some experimentation of this kind we discovered that a value of $b = 2.2$ works well when extracting strokes from low contrast images whose strokes often contain light sections (such as the qualitative examples shown in Section 4.2). For higher-contrast images such as the ICDAR contest images in Section 4.1 we found that a value of $b = 4$ works best. Although the quantitative examples were higher contrast, they had much tighter loops, so a higher b exponent was needed to prevent incorrectly jumping across gaps, especially since the strokes tended to be lightest at the apex of a loop.

In addition to interactively training the cost exponent b, we also created a method for Intelligent Pen to train itself. Since images in the same dataset or by the same author will often have strokes with about the same intensity we decided to let Intelligent Pen remember the pixel values of strokes it had seen and reduce the cost of pixels with commonly occurring intensities. To do this, Intelligent Pen maintains a histogram of the intensities of every stroke pixel it encounters. Whenever it extracts a new stroke it updates the histogram accordingly. We then scale the histogram to values between 0 and 1, giving a value $hist(I)$ for each possible intensity value. This scaled histogram is then used to adjust the cost function in Equation 3.1. Equation 3.5 shows how Equation 3.4 is modified to include the training data.

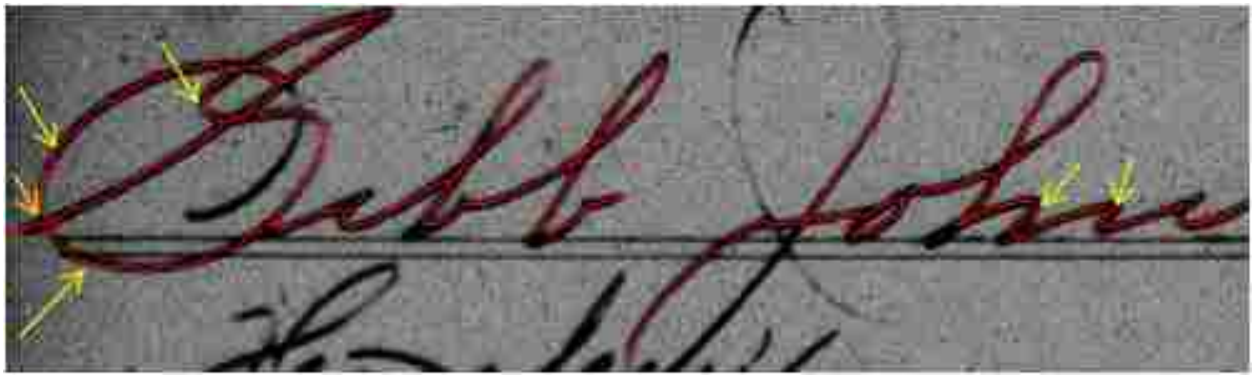
$$C_{xy} = i^b * (1 - hist(i)) + (0.75 * F_{xy} + 0.0001) \tag{3.5}$$

Because adding additional training data is simply a matter of incrementing values in the histogram we can easily update it as new data is collected. Additionally, we can save or load a trained model by simply exporting or importing the histogram counts. Figure 3.7 gives an example of how this training method can improve the algorithm's performance. Figure 3.7a shows the results without any prior training. Figure 3.7b shows how, after training, Intelligent Pen recovers strokes that it would have missed without training. As can be seen from the yellow arrows, this training process improves the accuracy as well as the completeness. One training pass eliminated several stubs where it started to search the background and sections where it recovered the same stroke twice. Though there is a false jump on the 'n' where there wasn't one before, this is a failure of the pruning process (discussed in Section 3.10), which is affected by the greater number of high-cost strokes recovered after training.

The methods described above, while effective in training an exponential cost function, failed to improve performance on the Sigmoidal function, and as such were ultimately discarded in favor of the untrained Sigmoidal cost function. However, there are still cases where they can be used to obtain improved results, as discussed below in Section 4.3.



(a) Without any prior training the algorithm misses a few important strokes.



(b) After one training pass the extracted strokes are more accurate and complete. Yellow arrows indicate some of the more subtle improvements.

Figure 3.7

3.5 Seed Point Selection

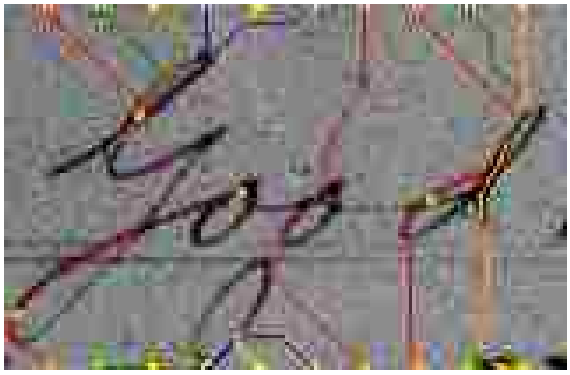
Now that we have our cost function and an algorithm for expanding a wavefront from a seed point, we need a good way to automatically choose what those seed points are. To do this, we choose several evenly spaced points on the top and bottom of the image, as in Figure 3.8a. We then use equations 3.1 and 3.2 to find the lowest-cost path from each top point to the bottom point directly below it. This is done by letting a wavefront continuously expand from the top point until it contains the bottom point, then following backpointers to get the shortest path.

Since each path must pass through any and all form lines we ignore the form line cost by setting F_{xy} to zero, saving computation time. Figures 3.8b - 3.8d show how the

(a) Points are chosen every 15 pixels along the top and bottom of the image (yellow). (160 x 104 pixels)

(b) The optimal path from each top point to its corresponding bottom point is found according to Equations 3.1 and 3.2.

(c) Seed point (orange) is center of lowest cost 5-pixel segment.



(d) Other seed points are found in the same way. Duplicate seed points or points within 25 pixels of previously found seed point are discarded.

Figure 3.8: Finding seed points.

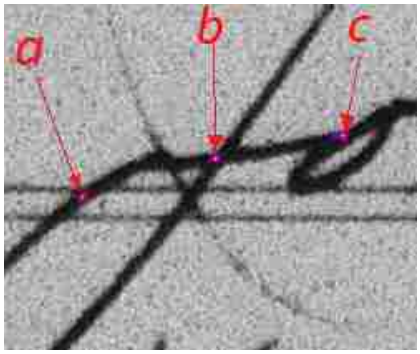
paths snap to any handwriting that is in proximity to the strip. For each path we then find the lowest-cost 5-pixel segment of the obtained path and choose the center of that low-cost segment as a seed point (Figure 3.8c). Because these seed points lie in the center of a low-cost area of the path it is likely that they lie on a stroke of handwriting. Since we're ignoring the form line cost F_{xy} we discard any seed points that are on or between form lines. Additional seed points are generated automatically (Figure 3.8d), so we require comparatively few seed points to initiate the process.

3.6 Parallel Path Expansion

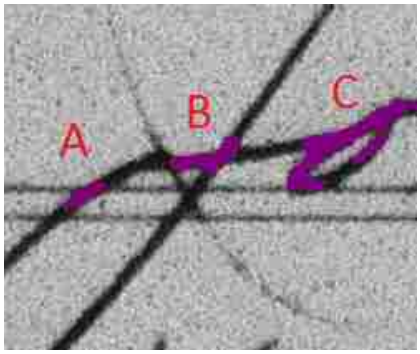
Thus far our examples have shown only a single wavefront at work. However, the method described above gives us multiple seed points. We allow each seed point to expand simultaneously (in parallel) in cost order, each one forming a wavefront. If for some reason a seed point is in a comparatively higher cost region of the image, it won't expanded as quickly as other seed points (Wavefront A in Figure 3.9b).

As shown in Figures 3.9c and 3.9d the wavefronts eventually collide and merge, then continue to expand until reaching the end of the strokes. Note that Intelligent Pen does not follow the horizontal lines because we set those lines to be higher cost since they are not part of the handwriting (see Figure 3.15b).

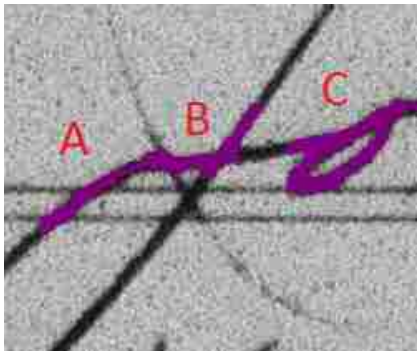
Since the goal is to operate on noisy strokes with gaps and/or low contrast sections, a key benefit of Intelligent Pen is that it does not simply do a hard form fill on connected components. Although a wavefront in a high-cost area may briefly stop expanding as other, lower-cost wavefronts are expanded, eventually the cumulative cost of the other wavefronts will be high enough that the higher cost wavefront gets expanded. As a result, when a wavefront reaches the end of a handwriting stroke it will eventually continue expanding past the edge of the stroke and into the lighter background pixels around it. This allows the algorithm to "jump" across gaps and light areas as in Figures 1.1 and 1.2.



(a) Seed Points a, b, and c associated with purple wavefronts A, B, and C. (249 x 209 pixels)



(b) Wavefront C is on a lower-cost stroke and expands faster than A and B.



(c) When wavefronts A and B meet, they merge and continue expanding.

(d) The unified wavefront continues to expand along the lowest-cost strokes.

Figure 3.9: Simultaneous Expansion of 3 Wavefronts A, B, and C.

3.7 Path Consensus

Because the wavefronts can continue to expand until they fill the image we need to decide when to stop expanding. Therefore, the wavefronts only expand until one of three stopping criteria is reached:

1. The number of pixels in the wavefront reaches a certain predefined threshold.
2. One wavefront runs into another actively expanding wavefront (Figure 3.9c).
3. A wavefront collides with the edge of the image.

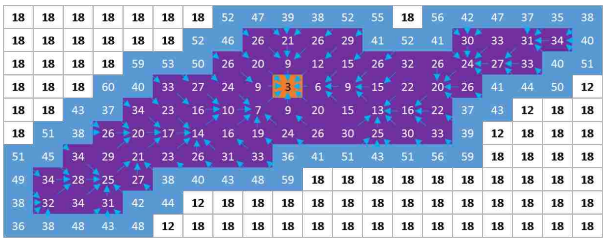
The reason for criteria 1 is that when the wavefront gets too large it becomes more susceptible to background noise, causing the algorithm to return more spurious path segments when obtaining consensus paths (as described below). It also becomes needlessly computational as the area of the wavefront increases $O(n^2)$ where n is the average width of the wavefront. Criteria 2 allows wavefronts to gracefully merge connecting path segments. Criteria 3 prevents spurious path segments from being found that otherwise come from expanding along the edge of the image.

Algorithm 2 Obtaining Consensus Paths from a Wavefront

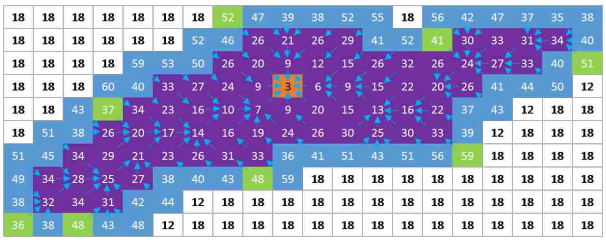
Input: Wavefront W with seed point S
Output: Set P of all consensus paths p

```
Find set B of each point on border of W
Create set F of every nth point in B
for f_i in F do
    Follow f_i's backpointers to get shortest path c_i to S.
for Each candidate path c_i do
    Find all points p in c_i that are also on another candidate path c_j // consensus paths
```

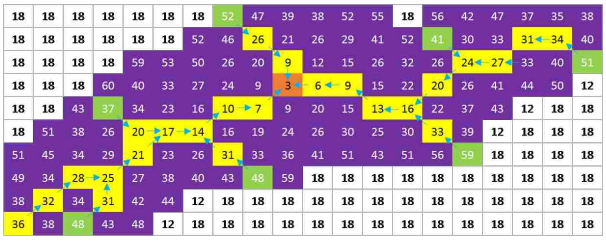
Figure 3.10 uses our earlier simplified example to demonstrate how locally optimal path segments are extracted from the wavefront using path consensus. First, we use a contour following algorithm to obtain an ordered list of all the points on the outside border of the wavefront (Figure 3.10a). Next, several "free points" (green in Figure 3.10b) are selected at regular intervals around this border. We then follow the backpointers from each free point to



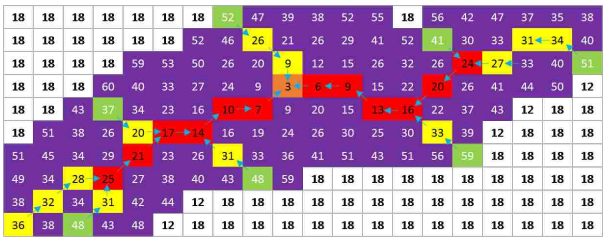
(a) An ordered set of border points (blue) is obtained by applying a contour following algorithm to the wavefront.



(b) Free points (green) are selected at regular intervals around the wavefront's border.



(c) For each of these free points the shortest path (yellow) to the seed point is found.



(d) Consensus points (red) appear on more than one of the yellow candidate paths and are used to define the optimal path for this segment.

Figure 3.10: Using path consensus to extract strokes (red pixels in (d)).

the seed point, which gives the shortest path for each free point (the yellow paths in Figure 3.10c). Points lying on two or more of these candidate paths are chosen as consensus paths (the red paths in Figure 3.10d). The red consensus points define the optimal path for this segment. This process is further outlined in Algorithm 2.

When a stopping event is caused by the collision of two wavefronts one additional step is taken. For each wavefront the path from the seed point to the point of intersection is also saved as a consensus point. This guarantees stroke continuity between wavefronts, as shown in Figure 3.11.

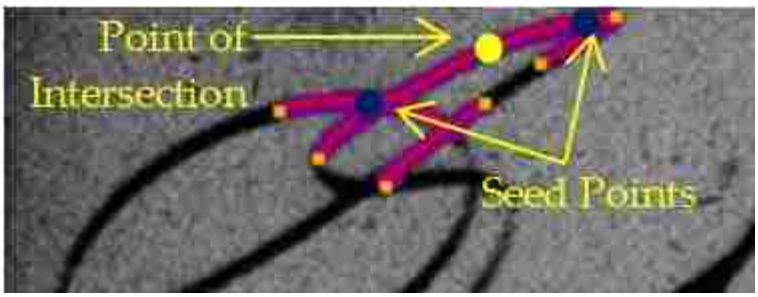


Figure 3.11: To ensure continuity we connect wavefronts at the point of intersection. (376 x 144 pixels)

3.8 Creating New Seed Points

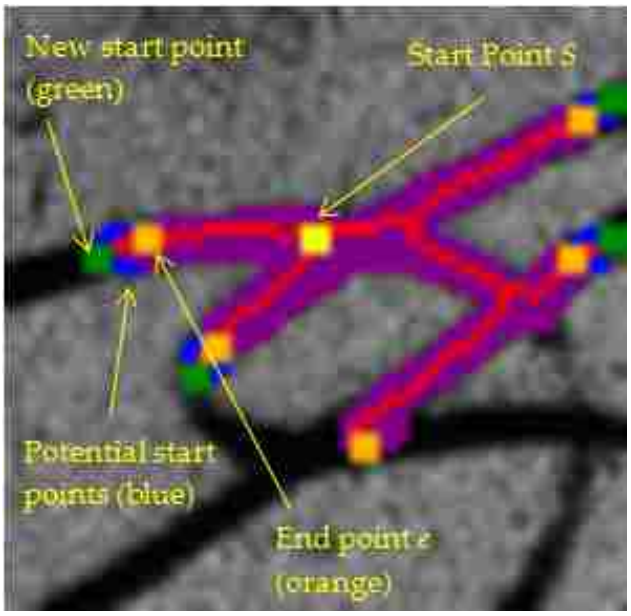
Having reached one of our wavefront stopping criteria, the next step is to generate a set of new wavefronts starting from new seed points. To do this we extend each of the newly acquired consensus paths to the edge of the wavefront, as shown in Figure 3.12 and Algorithm 3.

This is done by first finding the endpoint e of each consensus path (orange), then examining each point p_i on the outside edge of the wavefront that contains e in its path back to the seed point S (shown in blue in Figure 3.12). We then choose the p_i whose path from the seed point has the lowest cost per pixel. The consensus path is therefore extended to contain all of the path from the seed point to the chosen p_i (green), and p_i becomes a new seed point. This process is repeated for each consensus path.

Algorithm 3 Extending a start point to the edge of the wavefront

Input: Start point S, Consensus path with endpoint e.
Output: Extended path with endpoint e_2 on wavefront border

```
P ← all points on edge of wavefront
for p_i in P do
    Follow p_i's backpointers to get shortest path to S.
    if e is not on p_i's shortest path to S then
        Remove p_i from P
Order all p_i in descending order by length of shortest path to S
Choose first p_i (i.e. with longest path) as new start point.
Break ties by choosing p_i with lowest cumulative cost.
```



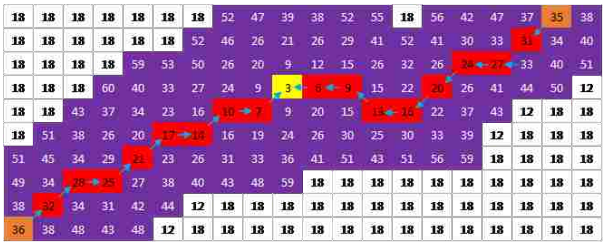
(a) Consensus path end point e (orange) is extended to the edge of the wavefront by choosing the point p_i that is the furthest point from S that has e on its shortest path.

18	18	18	18	18	18	18	52	47	39	38	52	55	18	56	42	47	37	35	38
18	18	18	18	18	18	52	46	26	21	26	29	41	52	41	30	33	31	34	40
18	18	18	18	59	53	50	26	20	9	12	15	26	32	26	24	27	33	40	51
18	18	18	60	40	33	27	24	9	3	6	9	15	22	20	26	41	44	50	12
18	18	43	37	34	23	16	10	7	9	20	15	18	16	22	37	43	12	18	18
18	51	38	26	20	17	16	16	19	24	26	30	25	30	33	39	12	18	18	18
51	45	34	29	21	23	26	31	33	36	41	51	43	51	56	59	18	18	18	18
49	34	28	25	27	38	40	43	48	59	18	18	18	18	18	18	18	18	18	18
38	32	34	31	42	44	12	18	18	18	18	18	18	18	18	18	18	18	18	18
36	38	48	43	48	12	18	18	18	18	18	18	18	18	18	18	18	18	18	18

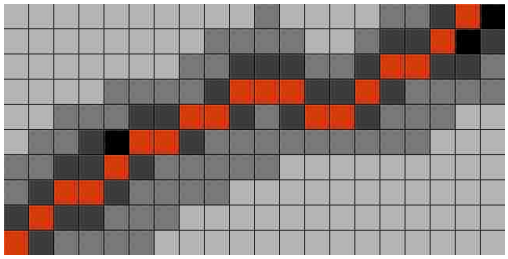
(b) Extending start points in our numerical example. Green pixels (end points) are part of, and complete the (red) optimal path for this image.

Figure 3.12: Extending Start Points to the Edge of the Wavefront

At this point our numerical example of wavefront expansion is complete. The final extracted stroke is shown in Figure 3.13 alongside the actual line. As can be seen, using an expanding wavefront for a depth-first search gives us a piecewise optimal path that corresponds directly with the darkest part of the stroke.



(a) The final path extracted by Intelligent Pen in our numerical example.



(b) The extracted stroke overlaid on the original.

Figure 3.13: Finalizing our Numerical Example

One drawback of this method for extending the path is that it can artificially extend false paths with only a few points of consensus that would otherwise have been discarded, or it can lead to consensus paths leaving a stroke and ending in the background. To prevent this, we apply the Otsu algorithm to obtain a threshold for the cost of pixels on each consensus path. We then work back from the end of each consensus point, removing any pixels that are above the obtained threshold, as in Figure 3.14.

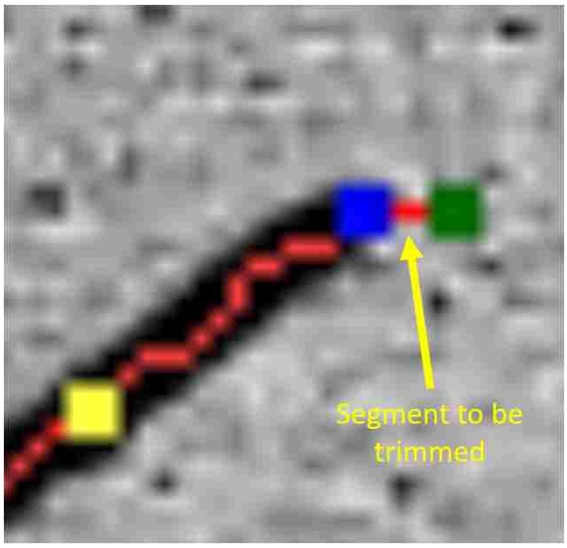
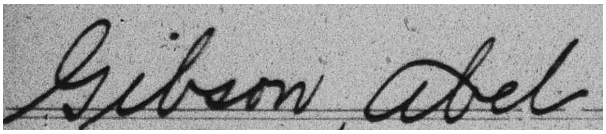


Figure 3.14: Extending the consensus path to the edge of the wavefront causes it to extend past the end of the stroke. It is trimmed back from the green point to the blue point by finding the Otsu threshold of the path and working back until the last point that is less than the threshold.

Having extended the consensus paths to the edge of the wavefront, we then choose the endpoints of each consensus path as seed points for new wavefronts and continue to expand

all wavefronts in cost order. Finally, every point within the wavefront is set to have infinite cost (Figure 3.15), guaranteeing that no point in the wavefront will be explored again. This is to prevent backtracking over the same section of handwriting multiple times. This can be safely done because of the piecewise-optimal nature of each path. Since each consensus path is an optimal path, we know that there are no additional useful paths to be found anywhere in the wavefront or they would already have been discovered. Setting costs within the wavefront high also saves computation. Figure 3.15 shows how the cost map changes over time as the wavefront expands.



(a) Original handwriting image. (1159 x 242 pixels)

(b) Initial Cost Map. Form lines set to high cost. Note high-cost area around the form lines.

(c) Wavefront after several timesteps (started from orange seed points).

(d) Cost map with wavefront points set to infinite cost.

(e) Wavefronts after several more expansions.

(f) Cost map continues to expand with wavefronts.

Figure 3.15: Cost map changes over time as different parts of image are explored/expanded.

Lastly, we note that when two wavefronts collide (stopping criteria 2), new seed points are generated for each wavefront. Additionally, the point of intersection is discarded from the set of new seed points (since it would otherwise lie between two areas of infinitely high cost and therefore have nowhere to expand). As described above and in Figure 3.11, the paths from each wavefront's seed point to the point of intersection are also saved as consensus paths to preserve continuity.

3.9 Stopping Path Growth

To save computational cycles and avoid exploring areas that have a low probability of handwriting we stop expanding a wavefront if it becomes roughly circular. When there is no handwriting near it the wavefront tends to expand in a circular pattern since all costs are roughly equal. Therefore, if a wavefront is circular it has a low probability of containing handwriting. To determine when to discard such circular wavefronts we calculate an inverse eccentricity value ecc^{-1} as the number of points in the wavefront (W) divided by the length L of the longest path within it (Equation 3.6).

$$ecc^{-1} = W/L \tag{3.6}$$

Wavefronts that are following a stroke of handwriting such as those in Figures 3.9 and 3.15 tend to be long and skinny, with a low ecc^{-1} value. On the other hand, Figure 3.16 shows an example of a (green) circular wavefront that was pruned because of its high ecc^{-1} value.

Figure 3.16: A wavefront at the end of a stroke (green) expands in a circle and is pruned. (53 x 59 pixels)

Since such circular wavefronts tend to occur at the end of a stroke it makes sense to prune them, and in practice this optimization improves the accuracy of Intelligent Pen, saving compute time and avoiding exploring areas of the image with no handwriting.

3.10 Path Pruning

As we go through the steps above we keep track of each consensus path we find. We then use three conditions to prune out false positives, as outlined in Algorithm 4. These conditions are the cost-per-pixel ($p_i.cpp$ in Algorithm 4), inverse eccentricity ($p_i.ecc$), and order expanded ($p_i.order$).

Algorithm 4 Pruning High Cost Paths

Input: List of consensus paths P, and weights $w_c, w_e, and w_o$

$i \leftarrow 0$
for p_i in P **do**
    $p_i.cpp \leftarrow \frac{p.cost}{p.length}$ // cost per pixel of path
    $p_i.ecc \leftarrow \frac{p.wavefront.count}{p.length}$ // eccentricity of p's wavefront
    $p_i.order \leftarrow i$ // order in which paths were found
    $i \leftarrow i + 1$
for p_i in P **do**
    $p_i.cpp \leftarrow normalize(p.cpp)$
    $p_i.ecc \leftarrow normalize(p.ecc)$
    $p_i.order \leftarrow normalize(p.order)$
    $p_i.pruningCost \leftarrow (w_c * p_i.cpp) + (w_e * p_i.ecc) + (w_o * p_i.order)$
$t \leftarrow getOtsuThreshold(paths)$
for p_i in P **do**
    if $p_i.pruningCost > t$ and p_i has >n% pixels below Otsu threshold **then**
        $P.remove(p_i)$

The cost per pixel is defined as the cumulative cost of the path divided by its length. The inverse eccentricity of a path (Equation 3.6) is used to penalize circular wavefronts (see section on Stopping Path Growth above). We use the order in which the paths were found as a third pruning criterion because the wavefronts are expanded in cost order, so the last ones to be explored are less likely to contain handwriting.

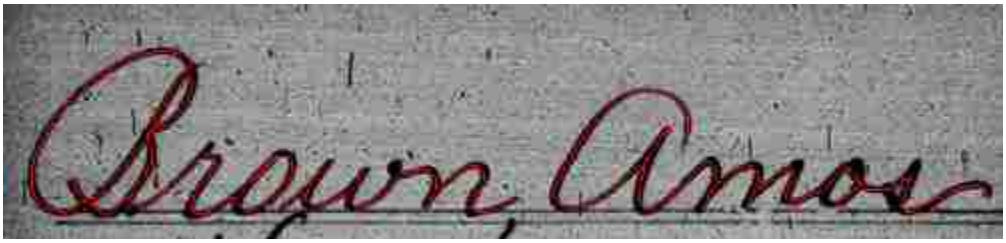
As outlined in Algorithm 4, once these three values are calculated for each path we normalize them, then use a weighted sum of the three to determine a final pruning cost. We can then apply Otsu thresholding to this set of pruning costs to remove any high-cost paths.

Figure 3.17 shows how Algorithm 4 is used to detect false positives. Figure 3.17a shows handwriting with all the strokes initially extracted by Intelligent Pen. The highest cost

(a) Each path is assigned a final cost based on cost per pixel, eccentricity, and the order found. High cost paths (blue) are removed using Otsu thresholding. (500 x 117 pixels)

(b) After using Otsu thresholding to remove high-cost strokes.

(c) By requiring a lower percentage of dark pixels on strokes between two other strokes we keep more valid strokes.



(d) Fewer high-cost strokes remain on the resulting image.

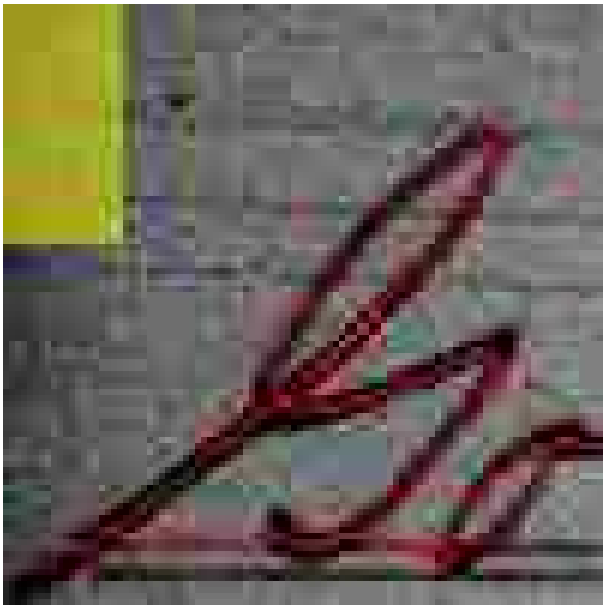
Figure 3.17: Pruning high-cost paths.

paths, marked in blue, are removed by Otsu thresholding, leaving the results shown in 3.17b. The qualitative examples shown in this document all used pruning weights of $w_c = 1$, $w_e = 0$, and $w_o = 0$. Although the examples shown in this work do not use $p_i.ecc$ and $p_i.order$, they are included here as examples of how the pruning algorithm could be extended to include

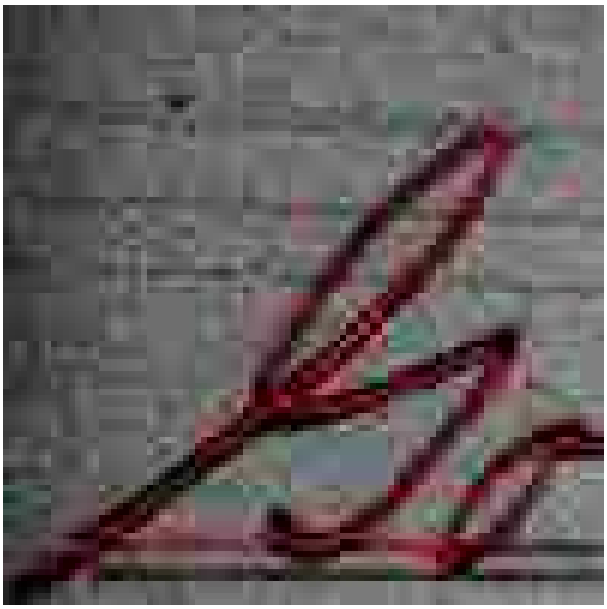
additional features. Future work will involve exploring what features would be most useful and automatically tuning feature weights for better results.

As an additional step, before pruning any line we check to see if more than n% of its pixels are below the Otsu threshold for the image. If not then they are kept. For strokes between two other strokes n is 75%, while for other strokes the value of n is 50%. This causes the algorithm to favor keeping paths that are between other low cost paths to avoid removing light strokes and other true positives, while high cost paths that are not between two other paths are less likely to be valid strokes. This leads to better retention of light strokes and strokes that cross high-cost areas such as form lines (Figures 3.17c and 3.17d).

Due to the nature of the datasets used we often encountered partial handwriting fragments on the edge of the image. Therefore as a final pruning step, we remove any short paths that are not centrally located. Specifically, any paths shorter than 35 points in length that lie entirely within 20 pixels of the edge of the image are removed. While this step is somewhat heuristic in nature it is needed due to the fact that individual cells in form images often contain stroke fragments from adjacent cells, and we don't want to include those in our final results. Figure 3.18 gives an example of how this measure works well for our purposes.



(a) Darkening at the edge of the image leads to line fragments. (100 x 100 pixels)



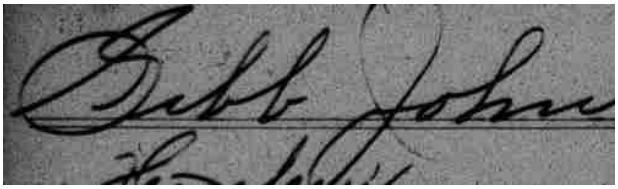
(b) By keeping only long or centrally located strokes we can prune the line fragments while preserving true positives.

Figure 3.18: Pruning line fragments on the edge of the image.

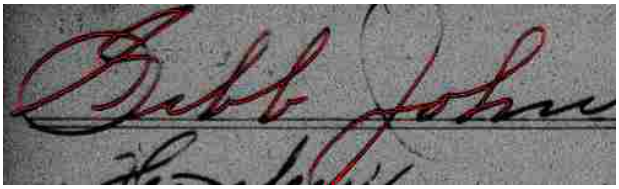
3.11 Additional Passes

In many cases one application of the algorithm will miss some portion of the text due to natural breaks in the handwriting, an incomplete set of seed points, etc. However, the algorithm can be repeatedly applied, with each iteration preserving information from the previous one. Each pass uses the ending cost map from the previous pass where wavefronts associated with previously explored costs are set high as in 3.15. This minimizes computation on subsequent passes by focusing the search only on unexplored regions.

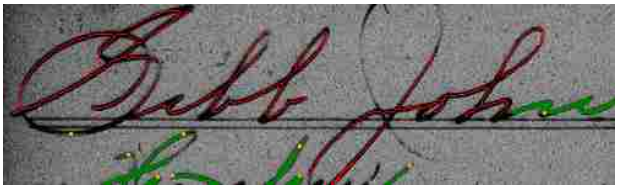
Figure 3.19 shows how we obtain seed points for additional passes. After the initial pass (Figure 3.19b), we look for unexplored handwriting by Otsu thresholding the image using only the unexplored points as input, then finding all the connected components of unexplored pixels (green strokes in Figure 3.19c). To reduce sensitivity to background noise only connected components containing more than 25 pixels are kept. For each connected component we find the lowest cost 7 x 7 pixel neighborhood that is not on the border of the



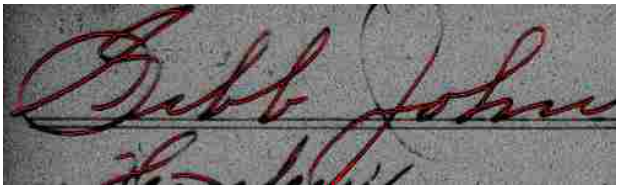
(a) Original Image. (500 x 147 pixels)



(b) The first pass misses a few strokes.



(c) Previously visited pixels are ignored on the second pass. New start points (orange) are the center of the 7 x 7 neighborhood with lowest cost in the unvisited connected components (green).



(d) Additional passes recover additional strokes, repeating until no new strokes are found.

Figure 3.19: Using additional passes to pick up missing strokes.

image, and use its center as a new start point. The final result, shown in Figure 3.19d, shows how the extra passes allowed us to recover some of the missing strokes.

3.12 Combining and Ordering Paths

Having obtained a set of consensus paths, we then group them together into larger connected components. In an ideal situation each connected component would represent an entire cursive word. For example, in 3.20, the name "Gibb" consists of two connected components: the 'G' (red) and the rest of the letters (green), while the name "John" is part of a separate connected component (lavender). It's fairly common for extra connected components to be created for capital letters that don't connect to the word. In some cases this also occurs from situations where wavefronts failed to connect properly (such as the base of the 'J' in "John").

Figure 3.20: Once the strokes are obtained they are joined together into connected components.

As a final step, Intelligent Pen performs a basic ordering of the pixels in each connected component. For each connected component we build an ordered set of pixels as outlined in Algorithm 5. We start by adding the leftmost pixel to a stack. While the stack is not empty we pop a pixel off the stack and add it to the ordered set, then see if there's a neighboring pixel not in the set already. If there's only one, we push it onto the stack and continue. If there are multiple neighbors not already in the stack or the ordered list then we know we're at an intersection, and we want to take the branch that continues in the same direction as the path we've been following. To do this we find the trajectory of the past 10 pixels on the path, then calculate the predicted position of the next point p_{i+1}. Then for each neighbor n_i we calculate the difference d_i between n_i and p_{i+1}. We then sort the neighbors by d_i, and

push them onto the stack in that order. As a result, the neighbor that changes direction the least is processed first, giving the stroke continuity as shown in 3.21.

Algorithm 5 Ordering Paths

Input: List of connected points C, empty list of points O, empty stack S, and integer d:

```
1: Sort C from left to right
2: Grab first (leftmost) point in C and push it onto S
3: while S is not empty do
4:     Pop point p off of S
5:     Add p to O
6:     Get all points N in C that neighbor p and are not in O or S
7:     if There is only one point n_i in N then
8:         Push n_i onto S
9:     else
10:        Fit polynomial function f to past n points
11:        for n_i in N do
12:            Find next n points on same path as n_i
13:            for Each of these n points p_i do
14:                d_i = p_i - f(p_i)
15:            Calculate sum d of all d_i
16:        Order N by d
17:        Push each n_i onto S
```

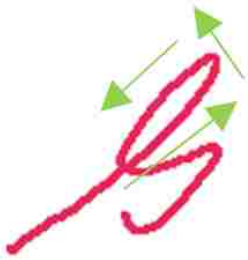


Figure 3.21: The strokes are ordered to correctly navigate junctions such as the loop in 'G'.

3.13 The Intelligent Pen Software Interface

While Intelligent Pen has been designed to be as general as possible, there are a few variables that need to be configured. Figure 3.22 shows the Intelligent Pen program interface, along with some of the variables that it allows the user to adjust.

Figure 3.22: Intelligent Pen's graphical user interface.

The image is displayed in the center-left of the window, along with the x and y coordinates of the cursor and the RGB values of the pixel under the cursor. To the right is displayed the Search Radius, Graph Search Step, Neighborhood size, and Path Search Interval, as well as the pruning weights described in Section 3.10. The search radius is used in determining the point at which a wavefront is large enough that we stop its growth, as described in Section 3.7. The Graph search step is used when finding seed points, and represents the distance between the yellow points in Figure 3.8. The neighborhood size is used to determine the local minimum of each path obtained when finding seed points (Figure 3.8b). Finally, the path search interval is the number of pixels between each free point when finding consensus paths, as in Figure 3.10b. There is also a checkbox that allows toggling whether or not to show the high cost paths that are pruned according to Section 3.10.

Chapter 4

Results

Two datasets were chosen to validate the effectiveness of the algorithm: one provided by an ICDAR contest[1], and one provided by FamilySearch. The ICDAR dataset provides a quantitative look at the algorithm's performance, while the FamilySearch Dataset provides several qualitative demonstrations of Intelligent Pen.

4.1 Quantitative Results

The first dataset examined was taken from a contest in connection with ICDAR 2013[1]. It consisted of 605 signatures that were acquired using a Wacom Intuos4 Large digitizing tablet and a Wacom Inking pen. The signatures were written on a blank paper on top of the tablet to acquire the online information, then the paper was scanned to provide the data in an offline format. For our analysis we used the 605 signatures that were provided with numeric ground truth data (x, y coordinates).

Because the signatures were captured in a clean environment using ink on a blank paper they lack several of the challenges common to historical documents such as background noise, form lines, and degradation. However, there are several examples of low-contrast, light strokes and minor gaps in the handwriting. For each image, Intelligent Pen was used to extract a series of pixels representing the strokes of the signature, marked in red in Figures 4.1 and 4.6.

The online data (ground truth) consists of a series of points representing the pen's trajectory, marked in orange in Figure 4.1a. As can be seen from this example, while the

(a) Sparse ground truth (orange). (499 x 457 pixels)

(b) Ground truth after snapping to line center

(c) Ground truth after connecting nearby points, snapping to the center, and reconnecting.

(d) Strokes extracted by Intelligent Pen

Figure 4.1: A Signature from the ICDAR Dataset.

ground truth follows the general path of the signature it does not completely align with the actual stroke. This is presumably due to parallax in the image capture method. Additionally, the points are sparse and do not provide a complete stroke. However, we decided to use this dataset despite these obstacles as it has been used previously as a benchmark with associated ground truth. To account for the poor alignment, ground truth points were snapped to the

center of the nearest stroke when making comparisons (Figure 4.1b). Adjacent points were then connected to avoid sparseness. As a final measure these connected points were snapped to the center again to make sure all the points were on the line center, then adjacent points were also connected again to avoid sparseness (Figure 4.1c).

Table 4.1 gives a summary of the results. In this case the precision measure is given by finding the average distance from each point on the strokes extracted by Intelligent Pen to the nearest ground truth point. Conversely, the average distance from each point on the ground truth stroke to the nearest point on the Intelligent Pen strokes is used as a recall measure. Figure 4.2 show the same data in graphical form.

	Precision	Recall
Mean Distance:	0.78	0.80
% within 0 pixels:	60.4%	49.8%
% within 1 pixel:	87.2%	84.7%
% within 2 pixels:	92.5%	92.6%
% within 3 pixels:	94.5%	96.0%
% within 4 pixels:	96.2%	97.4%
% within 5 pixels:	97.3%	98.5%

Table 4.1: Comparison of Ground Truth for the ICDAR Dataset with strokes extracted by Intelligent Pen. All numbers are averaged over all 605 signatures.

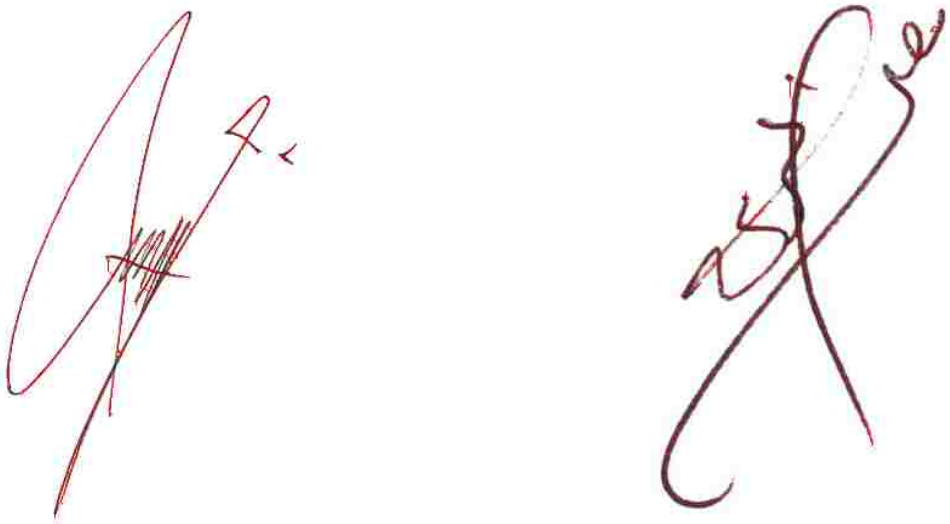
Figure 4.3, 4.4, and 4.5 give a few more examples of images from this dataset, along with the strokes extracted by Intelligent Pen (red). Figure 4.3 shows two cases where Intelligent Pen fails to recover some faint strokes.

In Figure 4.3a this is because there are multiple faint strokes close together, and a single wavefront expands across all of them. Because the strokes are faint and close together the optimal path leads to false bridges between the strokes, but because a wavefront already explored that area these strokes are missed on the second and third pass of the algorithm. A potential solution to this type of problem may be to use higher resolution images, though in this case we were limited to the images available.

Figure 4.2: A graphical representation of Table 4.1.

In Figure 4.3b the missed stroke is so light in comparison to the background that any wavefronts that start exploring it are pruned back due to high eccentricity. This is one difficulty that appeared repeatedly in the process of creating Intelligent Pen. In order to avoid false bridges between strokes, like those in Figure 4.3a, the cost function must be averse to crossing light sections of the image, leading to strokes such as the one in 4.3b being missed because they are so light or contain gaps. However, this leads to light strokes being missed. While Intelligent Pen is designed to address these issues, an ongoing challenge is finding the balance between recovering light strokes and bridging gaps versus creating false strokes and bridges. One approach to finding this balance is through interactive training to make the cost function data-specific (Section 3.13). In this specific case, while the individual pixels of the missed stroke are picked up on the second pass of the algorithm, no start points are placed there because the stroke is sparsely connected, with no single connected component having more than a few pixels in it. As such the algorithm mistakes these pixels for background noise and discards them.

Figure 4.4 gives some average cases with the occasional false bridges due to especially close lines in the original image. False bridges were the primary cause of error on the precision



(a) Intelligent Pen fails to recover some light strokes that are close together. (453 x 266 pixels)

(b) A faint stroke is not to explored because it does not contain large enough connected components. (500 x 492 pixels)

Figure 4.3: Intelligent Pen must balance an aversion to false bridges with an ability to recover true strokes that are very light, leading to some false positives and some missed strokes.

metric, while missing the occasional faint stroke was the main source of error in the recall metric. Figure 4.5 shows some examples where most of these pitfalls are avoided.

In some cases the ground truth data points (orange dots) were so poorly aligned with the image that even the after snapping to the line center and connecting the ground truth there were still obvious errors, as shown in 4.6. Because such poorly aligned images contributed to some of the error, 31 of them were removed from the dataset and the results were re-calculated. The resulting numbers are shown in Table 4.2.

As would be expected, this led to a slight improvement in the precision measure, while the recall measure stayed about the same (since the error was due to missing strokes in the ground truth the recall would not have been affected). While these improvements were minor due to being averaged across hundreds of images, the sub-pixel accuracy on both measures serves to demonstrate Intelligent Pen's ability to recover strokes with a high degree of confidence where the average difference between Intelligent Pen and the ground truth is less

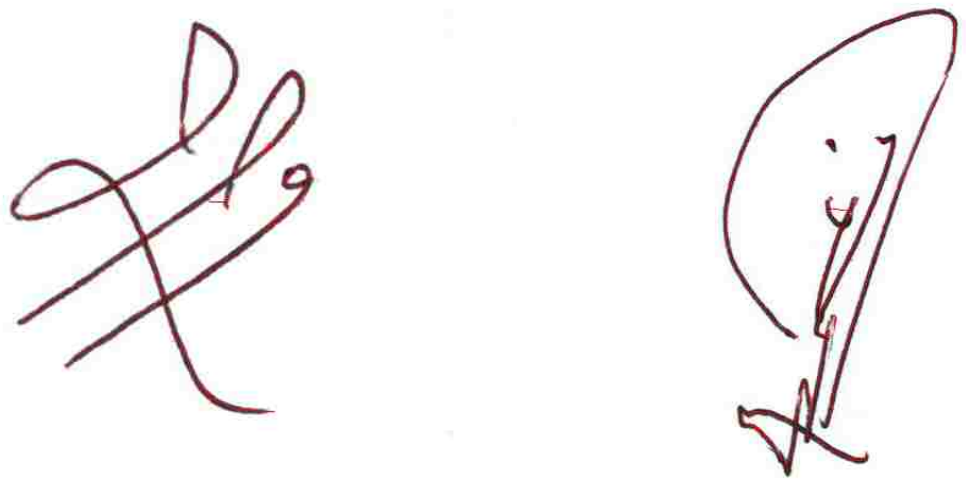


Figure 4.4: Some examples of average performance, where most of the strokes are recovered but there are a few false gaps or missing strokes. (Left: 500 x 444 pixels, Right: 335 x 500 pixels)

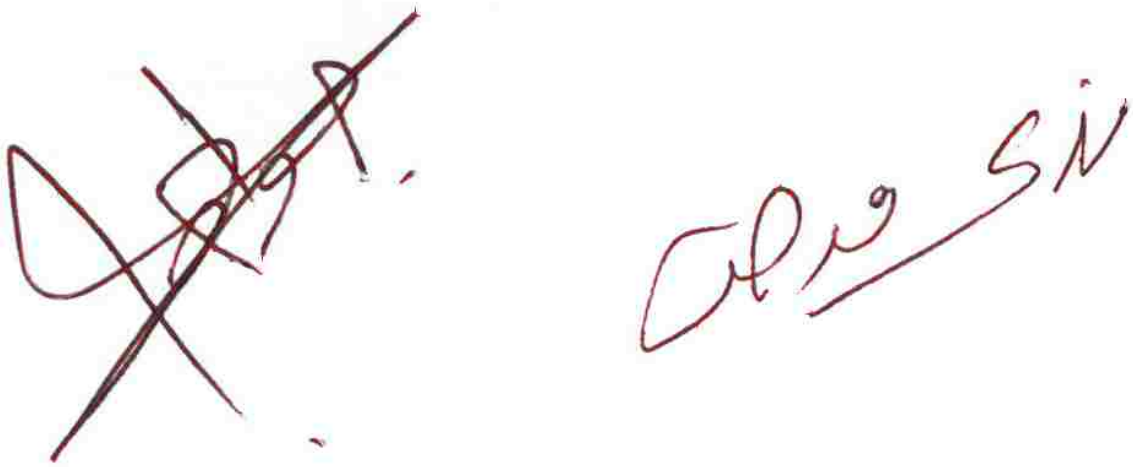
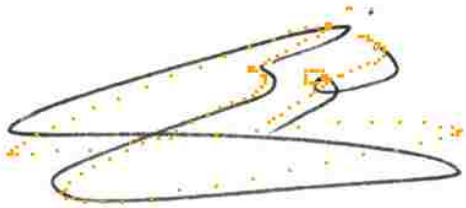
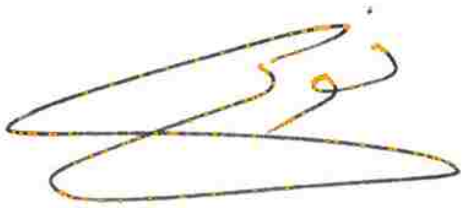


Figure 4.5: Some examples where Intelligent Pen produced especially clean and complete results. (Left: 500 x 449 pixels, Right: 499 x 275 pixels)

than a pixel. Further, in a handwriting recognition context, which is one of the motivating use cases of Intelligent Pen, a difference of 1-2 pixels should have little to no effect on the end recognition result.

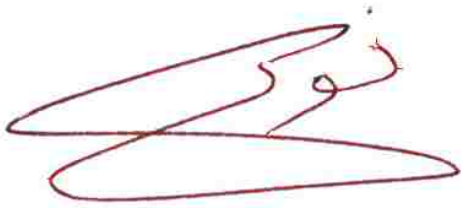


(a) Sparse ground truth (orange). (500 x 247 pixels)



(b) Ground truth after snapping to line center.

(c) The connected ground truth missed several pieces of the signature.



(d) Strokes extracted by Intelligent Pen.

Figure 4.6: A signature with poorly aligned ground truth.

	Precision	Recall
Mean Distance:	0.72	0.79
% within 0 pixels:	60.8%	50.0%
% within 1 pixel:	87.8%	85.1%
% within 2 pixels:	93.1%	92.8%
% within 3 pixels:	95.4%	96.2%
% within 4 pixels:	96.6%	97.5%
% within 5 pixels:	97.7%	98.5%

Table 4.2: Comparison of Ground Truth for the ICDAR Dataset with strokes extracted by Intelligent Pen after removing poorly aligned images.

4.2 Qualitative Results

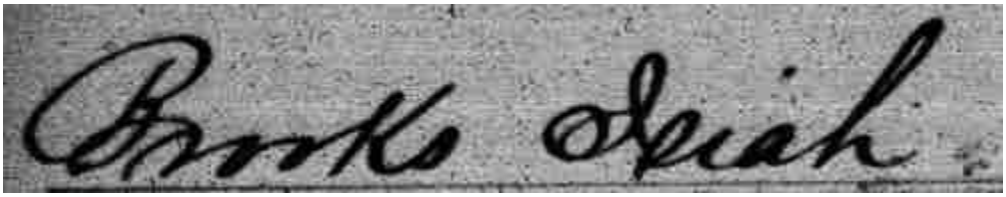
The second dataset is a set of cursive names from historical documents provided by FamilySearch. The images in this dataset all have noisy backgrounds, form lines cutting through the handwriting, ascenders and descenders. Many of the images have low contrast between the handwriting and the background, as well as gaps in the strokes and other imperfections in the writing. Figure 4.7 shows two pieces of handwriting from this dataset, along with the strokes obtained by Intelligent Pen after one pass. Figures 3.17 and 3.20 are also taken from the same dataset.

In Figure 4.7b, Intelligent Pen correctly recovers most of the strokes, but also picks up a false bridge between the 'B' and the 'r'. One interesting piece of the stroke is the gap at the top of the 'B', which intelligent pen recovers. Figure 4.7d Is a nearly perfect example except it fails to complete the circles on the 'o's and the 's'. Such failures can occur when a single wavefront covers the entire loop. Since there is generally only one optimal path around a looped stroke Intelligent Pen may not have sufficient consensus on all sides to fully recover the loop.

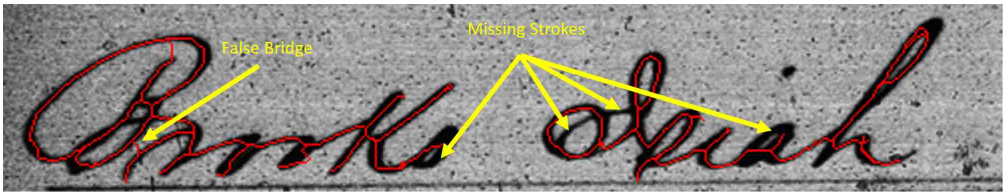
One key benefit of Intelligent Pen is its ability to recover ascenders and descenders while not including the form lines, due to the extra cost F_{xy} applied in 3.3. The J in Figure 4.8 is particularly successful, fully recovering the descender and ascender while not following the form lines or the additional noisy horizontal lines.

Figure 4.9 shows a few more examples. Figure 4.9b returns a fairly complete set of strokes, but misses a few strokes at the bottom of the 'b', e', 'a', and 'm'. This is likely due to those strokes having high-cost due to their proximity to the form line. The same is true for the jump across the bottom of the right half of the 'W'.

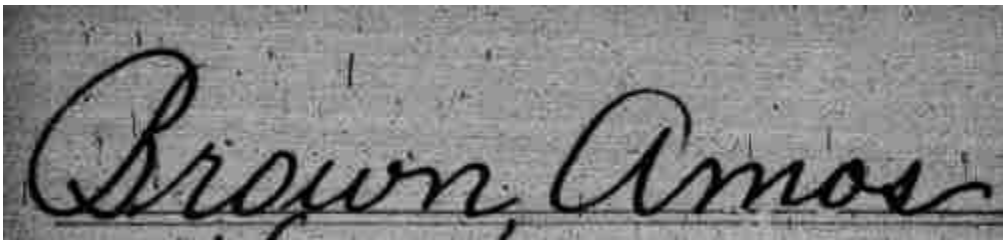
Figure 4.9d is a good example of Intelligent Pen's performance on thinner, lighter strokes. Where Figure 4.9b errs on the side of avoiding the form lines, Figure 4.9d has the opposite problem, and misses a few light connecting strokes between 's', 'o', and 'n' in favor of the form line. This happens because the form line cost F_{xy} is multiplied by 0.75 in Equation



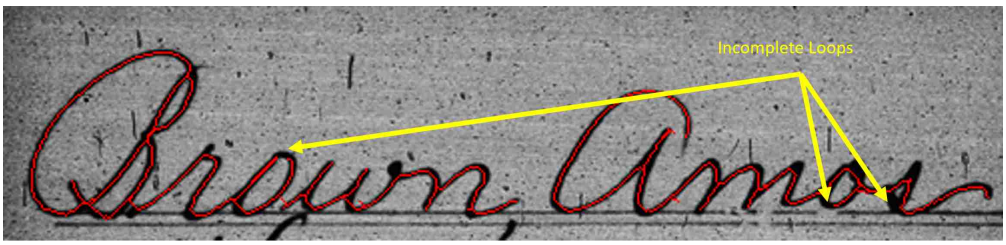
(a) 500 x 94 pixels



(b)



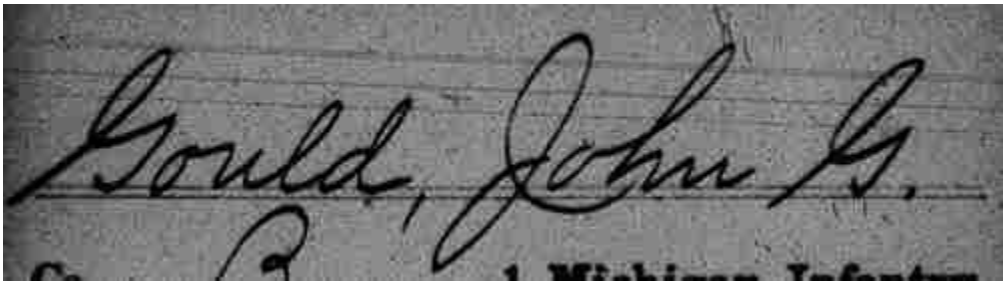
(c) 500 x 117 pixels



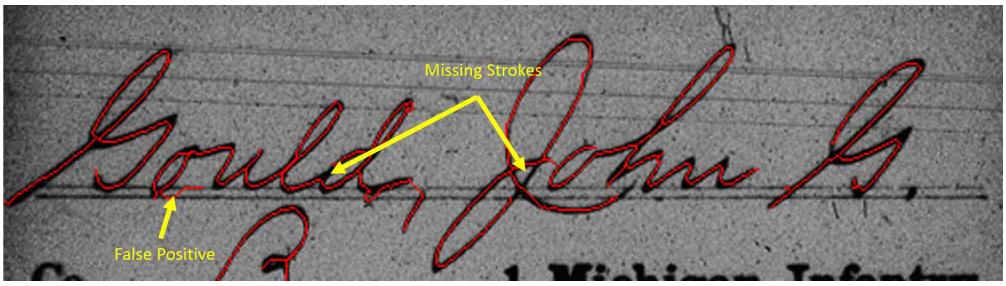
(d)

Figure 4.7: Examples of Intelligent Pen's performance on images in the FamilySearch dataset.

3.1, which means that exceptionally light strokes can, in some cases, be abandoned in favor of the form line. This is because our sigmoidal cost function climbs so steeply after the Otsu threshold t. As a result, light strokes that have an intensity 20 or so greater than t would have a cost of greater than 0.75, causing the form line to be preferred over the lighter connecting strokes in Figure 4.9d. However, increasing the form line cost aggravates the issues shown in Figure 4.9b. These two images are an example of some of the tradeoffs involved in tuning the



(a) 500 x 138 pixels

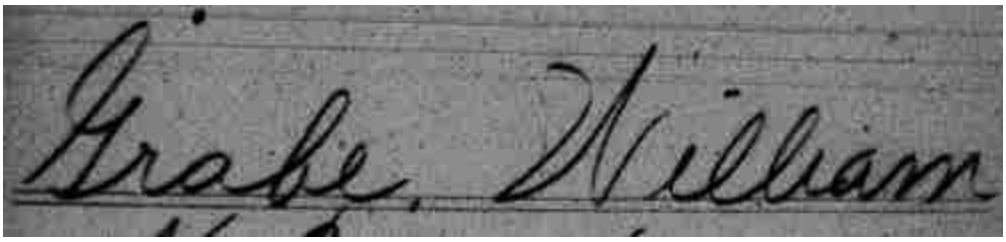


(b)

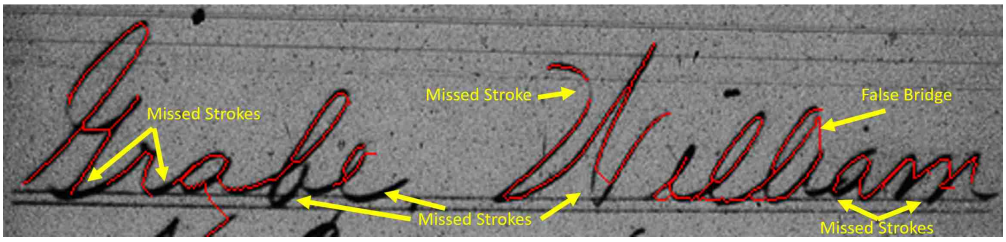
Figure 4.8: Recovering strokes in the presence of competing lines.

cost function. While the results are encouraging, we weren't able to find a cost function that could capture all strokes in all cases.

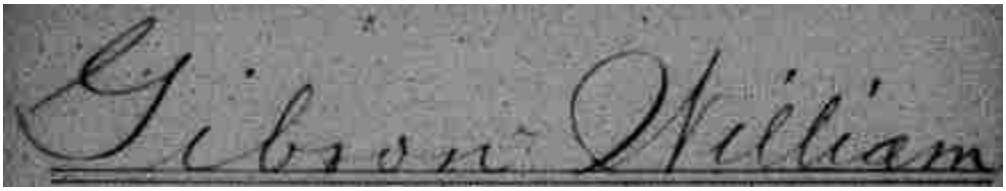
To evaluate the generality of Intelligent Pen we also tested it on some other historical documents, shown in Figure 4.10. While different from the FamilySearch images, these documents displayed many of the problems common to historical images. Figure 4.10b gets a mostly complete trace, though it fails to close the stroke on the first 'G' and the 'y', and misses the connector between 's' and 'b'. Figure 4.10d shows how Intelligent Pen is able to correctly identify the true strokes in the presence of false strokes bleeding through from the opposite side of the page. Figure 4.10f is a good example of extracting thinner strokes, missing only the faint stroke connecting 'x' and 'a'. There are also a few spurs that are the result of false positives that were pruned back, but only to the edge of the stroke rather than the center of the stroke (as described in Figure 3.14). Figure 4.10h shows another good stroke extraction, though it failed to catch the bottom part of the 'd' and part of the 'g' in "spangled". Overall



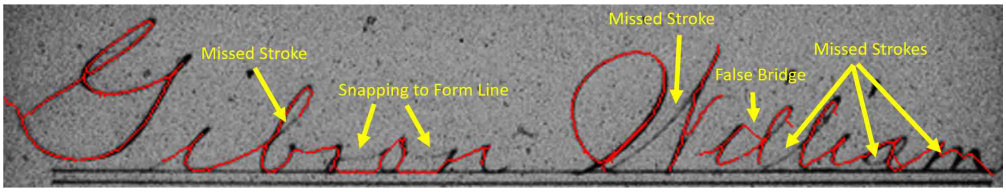
(a) 500 x 116 pixels



(b)



(c) 500 x 91 pixels



(d)

Figure 4.9: Some more examples of Intelligent Pen's performance on FamilySearch images.

Intelligent Pen's performance on these images is encouraging and demonstrates Intelligent Pen's ability to generalize well without needing additional configuration.

Another important context for handwriting extraction is historical journals. Figure 4.11 shows a piece of handwriting from a scanned journal of a Mormon missionary in Japan in 1910. Due to this particular example having thinner strokes than the other examples presented in this work the image had to be scaled up to get the results shown. This demonstrates the ability of Intelligent Pen to maintain reasonable robustness on lower resolution images even

(a) 431 x 133 pixels

(b)

(c) 388 x 127 pixels

(d)

(e) 411 x 233 pixels

(f)

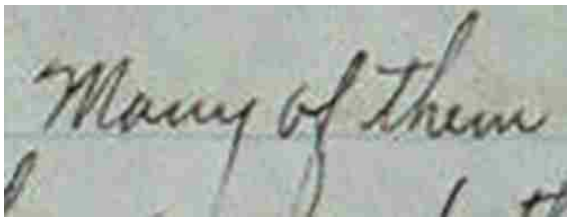
(g) 372 x 110 pixels

(h)

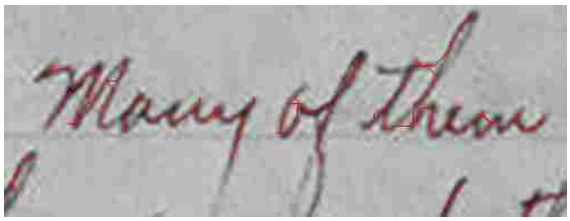
Figure 4.10: Intelligent Pen's performance on famous historical documents.

with JPEG artifacts. One could argue that these strokes would still provide adequate input to a handwriting recognition engine.

One of the benefits of Intelligent Pen is that it operates directly on the image's pixels and uses image features to extract strokes rather than language features. As a result, it's not tied to any specific language or character set. To demonstrate this we used Intelligent Pen to extract the strokes in Chinese handwriting, as shown in Figure 4.12. While some additional

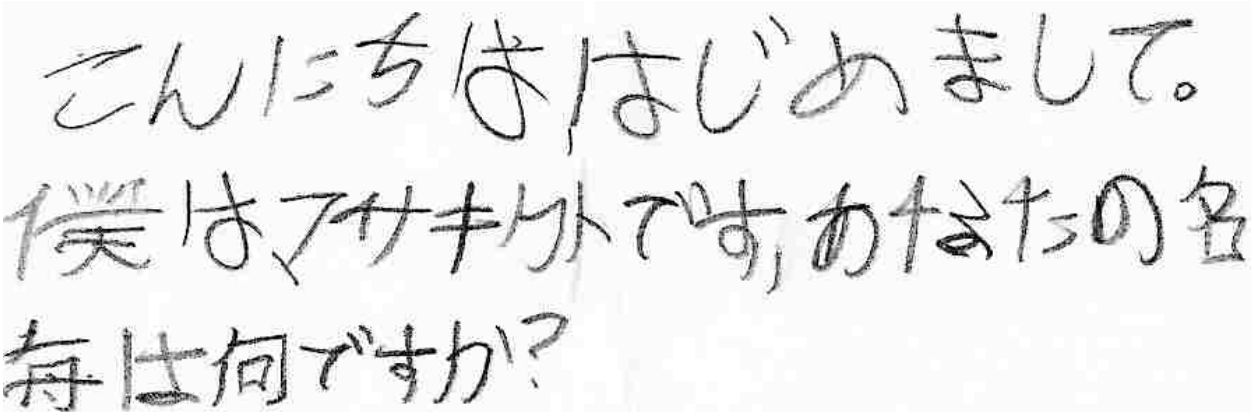


(a) Extract from the diary of a Mormon missionary in Japan. (500 x 188 pixels)

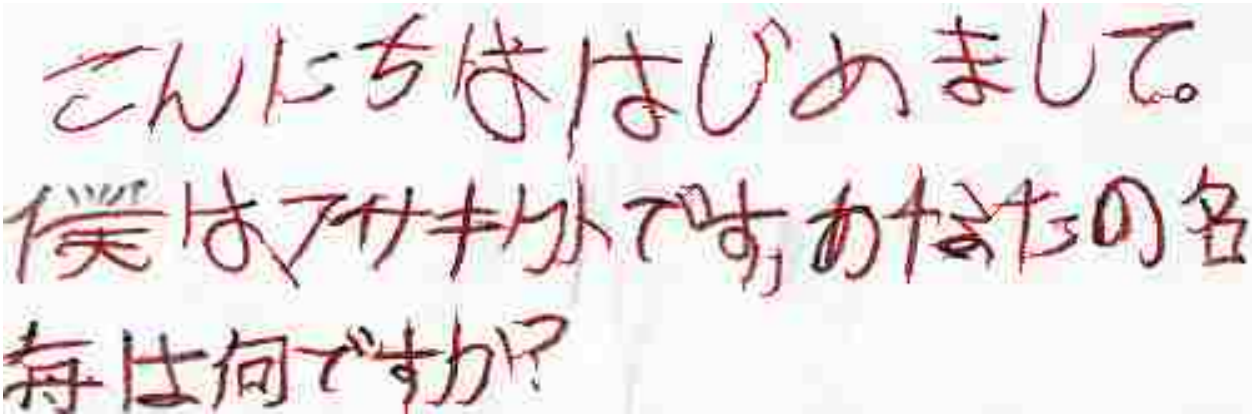


(b) Strokes extracted by Intelligent Pen.

Figure 4.11: An example of how poor resolution and JPEG artifacts can reduce result quality.



(a) Chinese Handwriting with variations in stroke intensity. (894 x 294 pixels)



(b) Intelligent Pen Recovers most of the strokes

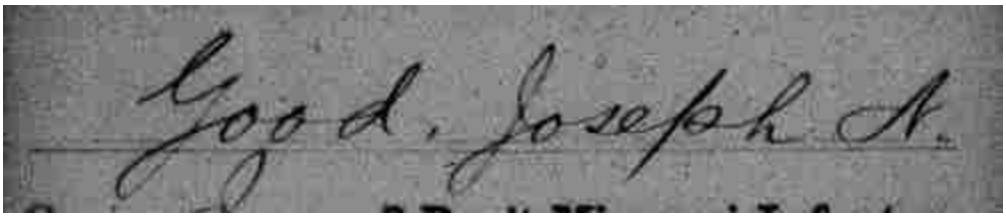
Figure 4.12: Intelligent Pen has the potential to be Language-Independent.

tuning to account for some of the smaller strokes may be needed, Intelligent Pen is able to recover nearly all of the strokes despite variations in stroke intensity. These results serve to show that Intelligent Pen has great potential for working with Asian characters, which is a tremendously important domain for historical image processing.

4.3 Analysis and Discussion

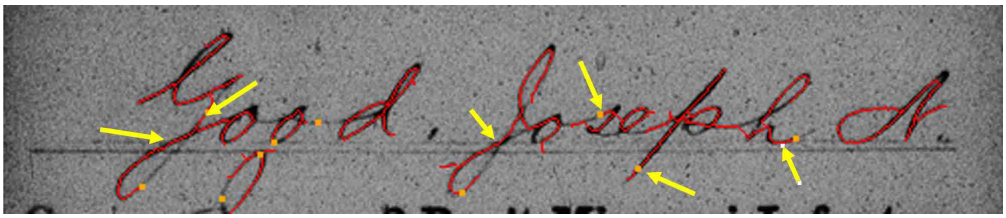
As can be seen from the images and numerical results above, Intelligent Pen is an algorithm with great potential that functions remarkably well over a wide variety of handwriting styles and quality. When it works well it works very well. However, there are a few issues where it is unable to find the right balance between recovering all the strokes and avoiding false positives. For example, even though we make additional passes until no unexplored strokes above the Otsu threshold can be found, there are still a few light strokes that don't get picked up, such as the connector between 's' and 'b' in Figure 4.10b or between 'x' and 'a' in Figure 4.10g. The reason for this is partly one of performance. We chose not to use a full top-to-bottom search of the image to find start points for second and third passes because the initial top-to-bottom search is one of the most computationally intensive pieces of the algorithm. By using the simpler Otsu-based method for finding the secondary passes the algorithm terminates in a fraction of the time. The cost of this is that the secondary iterations of the algorithm become subject to many of the same limitations as existing, thresholding-dependent techniques, with some chance of improvement due to the possibility of finding these faint strokes by expanding the wavefront from nearby dark strokes.

One of the key elements of Intelligent Pen is that it is a fully automatic system, whereas the original Intelligent Scissors was a user-guided system. As described above, Intelligent Pen uses the idea of path consensus to substitute for user-input and drive the automation process. However, it has also been shown that path consensus, in many instances, fails to recover all the strokes in an image. This can be due to the strokes being too faint to register, or simply a less than optimal choice of seed points. In Figure 4.13b we see an example where Intelligent Pen failed to recover all the strokes due to poor seed point placement. However, Figure 4.13c shows that by manually setting the seed points we can recover several strokes that were missed by the automatic process. This demonstrates that the failure is not with the minimum-cost path search method but instead in the way the algorithm is automated. It can be inferred that with further work the algorithm could be improved to recover more of



(a) An image with several faint strokes. (500 x 104 pixels)

(b) When automatically selecting seed points (orange), Intelligent Pen fails to recover several strokes.



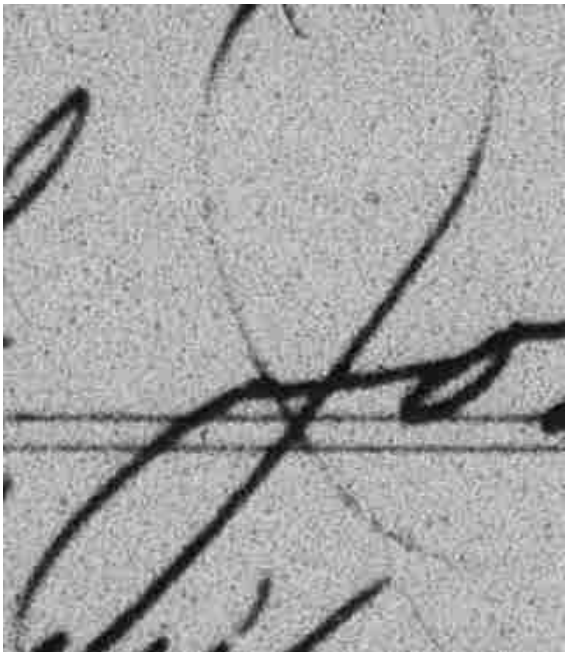
(c) Results for the same image after manually selecting seed points (orange). Additional strokes recovered are indicated with arrows.

Figure 4.13: Manually setting the seed points allows us to recover more strokes and demonstrates the better performance that could be achieved through better placement of seed points.

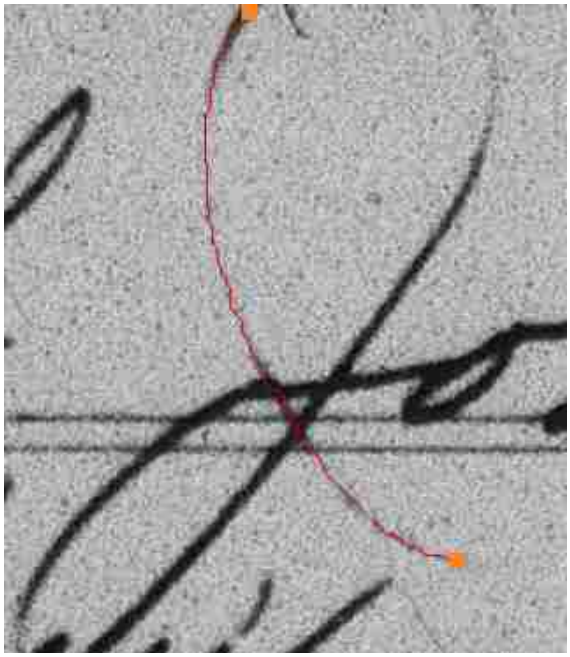
the missing strokes, from the use of more seed points, better seed point placement and better tuning of the cost function.

There are also some limitations to the sigmoidal cost function, in that it fails to recover some very faint strokes. In earlier experiments we used an exponential cost function, but eventually switched to the sigmoidal function because of its generally superior performance. However, there were some strokes, such as in Figure 4.14 that we could only detect by using an exponential cost function (as in Section 3.4) and finding the minimum-cost path between two manually defined points (the orange points in Figure 4.14b). The result is encouraging it

does show that the least-cost path search algorithm has the potential to recover more strokes if we can refine the placement of seed points and the automated stroke extraction process.



(a) An image with an extremely faint stroke. (320 x 369 pixels)



(b) Using an exponential cost function and finding the min-cost path between two points (orange) we can extract the stroke.

Figure 4.14: The potential for finding yet fainter strokes exists as we refine our method.

Chapter 5

Conclusion and Future Work

In summary, we have demonstrated that Intelligent has the ability to extract strokes from historical documents in the presence of noise, gaps, light strokes, and other difficulties. It draws inspiration from Intelligent Scissors in that it uses a minimum-cost path search function. This function is tuned to historical documents, and is automated rather than user-guided. To accomplish this automation, initial seed points are discovered by taking the local minima of the shortest paths from the top of the image to the bottom. Intelligent Pen then relies on path consensus to find strokes and automatically generate new seed points. Multiple cost wavefronts are expanded simultaneously, creating the potential for a parallel implementation. Stopping criteria are determined automatically, and, after removing previously explored sections of the image, additional passes of the algorithm are made to discover any previously missed strokes.

Unlike other handwriting extraction methods, which use some form of thresholding, Intelligent Pen operates directly on the grayscale images. We have demonstrated quantitatively that Intelligent Pen can extract handwriting strokes with a mean error of less than 1 pixel, and also demonstrated its performance qualitatively on several historical images from various contexts. Intelligent Pen was able to recover most of the strokes in the presence of noise, form lines, ascenders and descenders. In several instances a balance had to be struck between recall (finding all the strokes) and precision (avoiding false positives), but in general the method showed itself to be robust and viable for handwriting extraction.

While we have demonstrated Intelligent Pen's potential usefulness as an automated handwriting extraction tool, there are still improvements that could be made in future work. Although we have implemented a basic left-to-right method for ordering the strokes, the system could be improved by applying a more robust method to obtain the stroke ordering with which the handwriting was first created. This would be a key step towards using Intelligent Pen as a pre-processor for online handwriting recognition methods.

In addition, a better method for initializing the cost map for form lines is needed so that Intelligent Pen can more easily jump across form lines without attempting to follow them.

Another important line of inquiry would be to find the optimum placement of seed points on each iteration of the algorithm so that faint strokes can be recovered and no true strokes will be lost due to bad or incomplete placement of seed points or subsequent pruning. This could be further enhanced by finding a way to calculate the cost function so that faint strokes are preserved.

The sigmoidal cost function defined in Section 3.3 works well on a variety of images, but further experimentation could help discover a more robust cost function. As can be seen in Figure 3.3c, the sigmoidal cost function models the shape of the cumulative histogram, but is skewed towards the lower end of the histogram. More work is needed to determine whether results could be improved with a cost function that more closely aligns with the cumulative histogram of the image.

One other avenue that bears investigation would be to only set costs high for the consensus paths and one or two pixels to either side after exploring a wavefront. This has the potential to alleviate some of the missed strokes in images such as Figure 4.3.

Further refinement on Intelligent Pen's pruning methods would also be helpful. Currently the pruning algorithm (Algorithm 4) is only using the pixel intensity. As has been noted, the cost-per-pixel of a stroke as a pruning method has difficulty in differentiating between true strokes and false positives. More research is needed to determine if features

such as the order of discovery and the wavefront eccentricity can be used to determine better candidates for path pruning. Additionally, a more robust method for filtering out fragments on the edge of the image could improve the system. The current method of filtering out any short paths near the edge of the image is effective in practice but still something of a heuristic approach.

5.1 Final Thoughts

Although there are many avenues for extending Intelligent Pen, we have shown that in its current implementation Intelligent Pen is capable of overcoming many of the barriers to handwriting extraction in historical document images. Its sub-pixel accuracy on the ICDAR dataset shows excellent precision and recall. The results obtained on the FamilySearch images and other qualitative examples show its potential for opening doors to the challenging problem of handwriting recognition in older, noisy historical documents.

References

[1] ICDAR 2013 - Handwriting stroke recovery from offline data. https://www.kaggle.com/c/icdar2013-stroke-recovery-from-offline-data. [Online; accessed 13-June-2015].

[2] Impact: Improving access to text. http://www.impact-project.eu/about-the-project/concept/, 2015. [Online; accessed 6-October-2015].

[3] Paper to digital in 200+ languages. http://googleresearch.blogspot.com/2015/05/paper-to-digital-in-200-languages.html, 2015. [Online; accessed 6-October-2015].

[4] Transcriptorium project website. http://transcriptorium.eu/, 2015. [Online; accessed 6-October-2015].

[5] C.R. Allen and A. Navarro. The recognition of roman hand-written characters using dynamic information recovery. In *Image Processing and Its Applications, 1997., Sixth International Conference on*, volume 2, pages 741–745 vol.2, Jul 1997. doi: 10.1049/cp:19970994.

[6] G. Boccignone, A. Chianese, L. P. Cordella, and A. Marcelli. Recovering dynamic information from static handwriting. *Pattern Recognition*, 26(3):409–418, Mar 1993.

[7] Robert Clawson and William Barrett. Extraction of handwriting in tabular document images. *Family History Technology Workshop at Rootstech*, 2012.

[8] Robert Clawson, Kevin Bauer, Glen Chidester, Milan Pohontsch, Douglas Kennard, Jongha Ryu, and William Barrett. Automated recognition and extraction of tabular fields for the indexing of census records, 2013.

[9] H. Daher, D. Gaceb, V. Eglin, S. Bres, and N. Vincent. Ancient handwritings decomposition into graphemes and codebook generation based on graph coloring. In *Frontiers in Handwriting Recognition (ICFHR), 2010 International Conference on*, pages 119–124, Nov 2010. doi: 10.1109/ICFHR.2010.25.

[10] D.S. Doermann and Azriel Rosenfeld. Recovery of temporal information from static images of handwriting. In *Computer Vision and Pattern Recognition, 1992. Proceedings*

CVPR '92., 1992 IEEE Computer Society Conference on, pages 162–168, Jun 1992. doi: 10.1109/CVPR.1992.223211.

[11] FamilySearch. Familysearch home page. `https://familysearch.org/indexing/`, 2015. [Online; accessed 1-Sept-2015].

[12] Jake Gehring. personal communication, 2015.

[13] Tong Huang and Makoto Yasuhara. A total stroke slalom method for searching for the optimal drawing order of off-line handwriting. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, volume 3, pages 2789–2794 vol.3, Oct 1995. doi: 10.1109/ICSMC.1995.538204.

[14] P.M. Lallican and C. Viard-Gaudin. A kalman approach for stroke order recovering from off-line handwriting. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, volume 2, pages 519–522 vol.2, Aug 1997. doi: 10.1109/ICDAR.1997.620553.

[15] S. Lee and J.C. Pan. Offline tracing and representation of signatures. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(4):755–771, Jul 1992. ISSN 0018-9472. doi: 10.1109/21.156588.

[16] Eric L'Homer. Extraction of strokes in handwritten characters. *Pattern Recognition*, 33(7):1147 – 1160, 2000. ISSN 0031-3203. doi: http://dx.doi.org/10.1016/S0031-3203(99)00103-X. URL `http://www.sciencedirect.com/science/article/pii/S003132039900103X`.

[17] Eric N Mortensen and William A Barrett. Interactive segmentation with intelligent scissors. *Graphical models and image processing*, 60(5):349–384, 1998.

[18] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Systems, Man and Cybernetics, IEEE Transactions on*, 9(1):62–66, Jan 1979. ISSN 0018-9472. doi: 10.1109/TSMC.1979.4310076.

[19] Theo Pavlidis. A vectorizer and feature extractor for document recognition. *Comput. Vision Graph. Image Process.*, 35(1):111–127, July 1986. ISSN 0734-189X. doi: 10.1016/0734-189X(86)90128-3. URL `http://dx.doi.org/10.1016/0734-189X(86)90128-3`.

[20] Yu Qiao and M. Yasuhara. Recovering dynamic information from static handwritten images. In *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, pages 118–123, Oct 2004. doi: 10.1109/IWFHR.2004.87.

[21] Yu Qiao and Makoto Yasuhara. Recovering drawing order from offline handwritten image using direction context and optimal euler path. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 2, pages II–II, May 2006. doi: 10.1109/ICASSP.2006.1660455.

[22] Yu Qiao and Makoto Yasuhara. Recover writing trajectory from multiple stroked image using bidirectional dynamic search. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 2, pages 970–973, 2006. doi: 10.1109/ICPR.2006.984.

[23] Jun Tan, Jianhuang Lai, Wei-Shi Zheng, and C.Y. Suen. A novel approach for stroke extraction of off-line chinese handwritten characters based on optimum paths. In *Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*, pages 786–790, Sept 2012. doi: 10.1109/ICFHR.2012.165.

[24] Christian Viard-Gaudin, Pierre-Michel Lallican, and Stefan Knerr. Recognition-directed recovering of temporal information from handwriting images. *Pattern Recognition Letters*, 26(16):2537 – 2548, 2005. ISSN 0167-8655. doi: http://dx.doi.org/10.1016/j.patrec.2005.04.019. URL `http://www.sciencedirect.com/science/article/pii/S0167865505001674`.

[25] Wikipedia. Bellman equation — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Bellman_equation#Bellman.27s_Principle_of_Optimality`, 2015. [Online; accessed 21-May-2015].

[26] Chen Yan and G. Leedham. Decompose-threshold approach to handwriting extraction in degraded historical document images. In *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, pages 239–244, Oct 2004. doi: 10.1109/IWFHR.2004.33.