Geoinformatica
DOI 10.1007/s10707-014-0203-9

Recognizing text in raster maps

Yao-Yi Chiang · Craig A. Knoblock

Received: 15 December 2011 / Revised: 18 December 2013 / Accepted: 8 January 2014

Abstract Text labels in maps provide valuable geographic information by associating place names with locations. This information from historical maps is especially important since historical maps are very often the only source of past information about the earth. Recognizing the text labels is challenging because heterogeneous raster maps have varying image quality and complex map contents. In addition, the labels within a map do not follow a fixed orientation and can have various font types and sizes. Previous approaches typically handle a specific type of map or require intensive manual work. This paper presents a general approach that requires a small amount of user effort to semi-automatically recognize text labels in heterogeneous raster maps. Our approach exploits a few examples of text areas to extract text pixels and employs cartographic labeling principles to locate individual text labels. Each text label is then rotated automatically to horizontal and processed by conventional OCR software for character recognition. We compared our approach to a state-of-art commercial OCR product using 15 raster maps from 10 sources. Our evaluation shows that our approach enabled the commercial OCR product to handle raster maps and together produced significant higher text recognition accuracy than using the commercial OCR alone.

Keywords GIS · OCR · Raster maps · Text recognition · Map processing

Y.-Y. Chiang (✉)
Spatial Sciences Institute, University of Southern California, 3616 Trousdale Parkway, AHF B55, Los Angeles, CA 90089, USA
e-mail: yaoyic@usc.edu

C. A. Knoblock
Department of Computer Science, Information Sciences Institute, and Spatial Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, CA 90292, USA
e-mail: knoblock@isi.edu

1 Introduction

Text labels in raster maps link place names to geographic locations. Converting the text labels in a raster map to machine-editable text helps produce geospatial knowledge for understanding a map region. For example, the recognized text labels from historical maps can be used to generate gazetteers that contain geospatial information in the past; such information is not easily accessible from other sources and is valuable to many applications and research fields, such as social science research. Further, with the techniques to extract road information from raster maps [7], we have shown that the recognized text can be used to create a set of named road vector data [9] that can be used in a Geographic Information System (GIS). Finally, if a raster map is registered with other geospatial data, the recognized map text can be used for labeling and retrieval of the other geospatial data. For example, after a map is aligned with imagery using the unique road-intersection patterns [3, 5], we can search for specific objects/regions in the imagery using map labels, such as names of schools, lakes, and roads.

In classic text recognition systems, including most commercial optical character recognition (OCR) products, the first step is "zoning," which analyzes the layout of an input image for locating and ordering individual text blocks containing homogeneous text lines of the same orientation (i.e., zones) [15, 19, 20]. Next, each of the identified text blocks is processed for text recognition. However, this zoning approach cannot handle documents that do not have homogeneous text lines, such as artistic documents, pictorial images with text, engineering drawings, and maps [23]. For example, Fig. 1 shows an example map with multi-oriented text lines of multi-sized characters where text blocks with homogeneous text lines do not exist.

In the fields of image processing, pattern recognition, and graphics recognition, where OCR techniques are actively studied, maps are generally handled as a special type of input images and hence are very often ignored; or the techniques developed for maps are ad-hoc and focus on specific types of maps (see the related work section for details). For example, to recognize the non-homogeneous text (i.e., multi-oriented, multi-sized, and curved text), one line of research works on specific cases of non-homogeneous text, such as straight text lines [12, 26] and multi-oriented but similar-sized characters [12, 14]. However, these specific cases represent only a small fraction of the text labels in common maps. A more general approach is to recognize individual characters separately [1, 10], such as utilizing rotation invariant features of specific character sets for character recognition [10]. Nevertheless, this

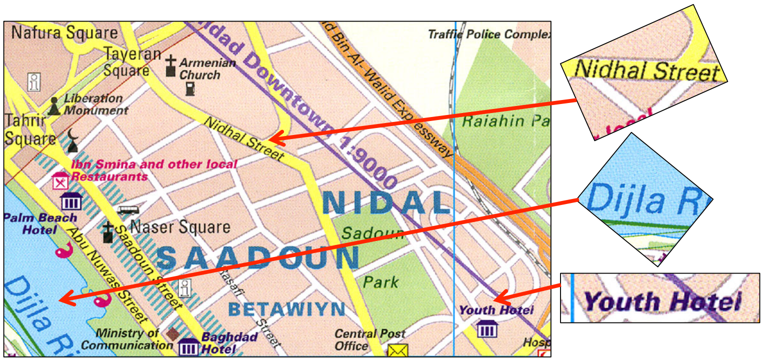


Fig. 1 Multi-oriented and multi-sized characters in a scanned Baghdad map by Gecko Maps

approach requires specific training work and cannot be easily integrated with the classic, well-developed OCR techniques, which process homogeneous text. Moreover, recognizing individual characters separately fails to take advantage of cartographic labeling principles and word context, such as utilizing a dictionary to help recognize grouped characters that represent meaningful words.

In this paper we present an end-to-end approach that reduces the amount of manual input to recognize text labels from full-size raster maps. Our techniques do not assume a particular series of maps or map source, which is achieved by incorporating cartographic labeling principles, such as that character spacing is generally smaller than string spacing [11] to generate inputs that can be used in a standard commercial OCR system. Figure 2 shows the two major steps of our approach: **(1) Text Layer Extraction** and **(2) Text Label Recognition**. The **Text Layer Extraction** step is a supervised technique that analyzes example text areas to identify colors that represent text in a raster map for separating individual text layers (i.e., a set of text pixels of the same color) from the map. The supervised technique does not require intensive user effort for handling a variety of raster maps, including scanned maps.

The **Text Label Recognition** step builds on the map processing work in our earlier papers that solved specific sub-problems for text recognition from maps (the detection of text orientations [6] and recognition of non-homogenous text [7]). This step handles multi-oriented, multi-sized, and curved text, requires no training for specific fonts, and can be easily integrated with a commercial OCR product for processing documents that contain non-homogeneous text. The **Text Label Recognition** step described in this paper offers a number of additional contributions beyond the integration of our previous text recognition techniques. First, we present the complete algorithms for the detection of text orientations [6] and recognition of non-homogenous text [7] (Sections 4). Second, we present a new technique for processing full-size raster maps (Section 4.1.4). Last, we integrate the **Text Layer Extraction** and **Text Label Recognition** steps to build a complete system for label recognition in maps.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the **Text Layer Extraction** step. Section 4 explains the **Text Label Recognition** step. Section 5 reports on our experimental results, and Section 6 presents the conclusion and future work.

2 Related work

Text recognition from documents that contain non-homogeneous text, such as from raster maps [22], is a difficult task, and hence much of the previous research only works on specific cases.

Fletcher and Kasturi [12] utilize the Hough transform to group characters and identify text strings. Since the Hough transform detects straight lines, their method cannot apply to curved strings. Moreover, their work does not handle multi-sized characters. Chen and Wang [4] utilize the Hough transform and a set of font and size independent features to recognize the numeric strings in raster maps. Their approach handles multi-sized numeric characters but cannot work on alphabetic characters.

Velázquez and Levachkine [30] and [25] present text recognition techniques based on detecting straight-string baselines for identifying individual text strings. Their techniques handle characters in various font sizes, font types, and orientations, but cannot work on curved strings.

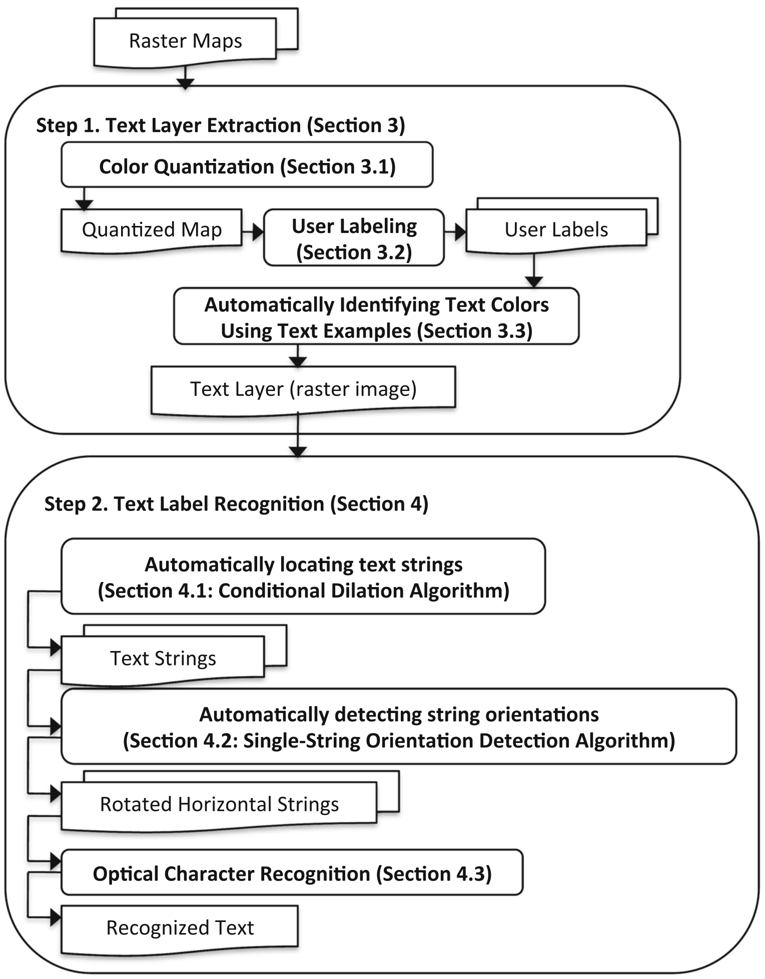

Raster Maps
Step 1. Text Layer Extraction (Section 3)
Color Quantization (Section 3.1)
Quantized Map
User Labeling (Section 3.2)
User Labels
Automatically Identifying Text Colors Using Text Examples (Section 3.3)
Text Layer (raster image)
Step 2. Text Label Recognition (Section 4)
Automatically locating text strings (Section 4.1: Conditional Dilation Algorithm)
Text Strings
Automatically detecting string orientations (Section 4.2: Single-String Orientation Detection Algorithm)
Rotated Horizontal Strings
Optical Character Recognition (Section 4.3)
Recognized Text

Fig. 2 The overall approach to recognizing text in raster maps

Goto and Aso [14] present a text recognition technique to handle multi-oriented and curved text strings. They first divide the input document into columns of equal sizes. Then the columns are divided into blocks based on the sizes of the connected components within each column to compute the local linearity of connected components and extract text strings. This approach works on touching characters, but requires characters of similar sizes.

In contrast to the previous research that works on specific cases [4, 12, 14, 25, 30], our approach processes heterogeneous raster maps using an interactive, training-by-example step (the **Text Layer Extraction** step) and handles multi-oriented, multi-sized, and curved text which commonly exists in raster maps (the **Text Label Recognition** step).

Li et al. [17] and [2] developed text recognition techniques that work on binary maps (i.e., bi-level map images) by assuming the map background can be easily removed manually. These text recognition techniques that work on binary maps cannot process scanned maps easily since scanned maps usually suffer from compression and scanning noise, which means that generating the input binary map requires tedious manual work. In comparison, our technique handles a variety of raster maps and requires only a few user interaction steps (the **Text Layer Extraction** step).

Pouderoux et al. [26] present a text recognition technique for raster maps. They first identify text strings in a map by analyzing the geometric properties of individual connected components in the map and then rotate the identified strings horizontally for OCR.

Roy et al. [28] detect text lines from multi-oriented, straight or curved strings. Their algorithm handles curved strings by applying a fixed threshold on the connecting angle between the centers of three nearby characters. This orientation detection method only allows a string to be classified into one of the four directions.

In the work of [26] and [28], their methods are based on the assumption that the string curvature can be accurately estimated from the line segments connecting each character center in a string. However, this assumption does not hold when the string characters have very different sizes in height or width. In contrast, we present a robust technique to estimate the curvature and orientation of a text string and our technique is independent from the character size (the **Text Label Recognition** step).

Deseilligny et al. [10] and [1] develop specific character recognition techniques for recognizing multi-oriented text strings. Deseilligny et al. [10] use rotation-invariant features and [1] use image features based on the Fourier-Mellin Transformation to compare the target characters with the trained character samples for text recognition. This type of technique requires intensive user effort for generating training data, such as providing sample characters for maps using different fonts to generate distinct feature sets for the classification. In comparison, our technique leverages the recognition task of a commercial OCR product and requires no user training for specific fonts (the **Text Label Recognition** step).

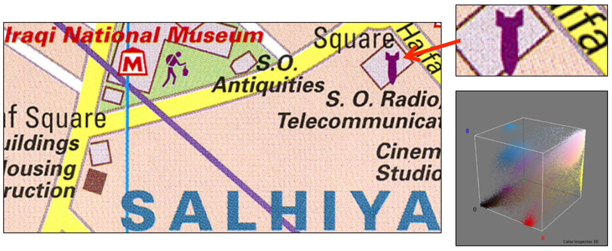
Other techniques introduce additional information for identifying text labels in the raster map. Gelbukh et al. [13] extend the algorithm from [30] by exploiting additional information from toponym databases and linguistic dictionaries. Myers et al. [21] generate hypotheses of the possible location and characters of text labels using a gazetteer. Our approach does not rely on the auxiliary information (e.g., a gazetteer), which is not available for many regions, especially for historical maps.

3 Text layer extraction

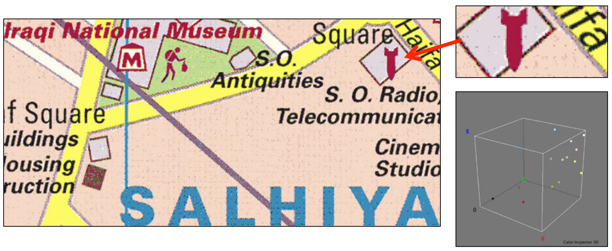
The **Text Layer Extraction** step is a supervised approach for extracting text layers from heterogeneous raster maps with varying image quality. This supervised approach first generates a quantized map where the text labels are represented by only a few colors. Next, a user provides examples of text labels in the quantized image. Finally, with the text examples, a set of text colors is automatically identified to extract individual text layers from the raster map.

3.1 Color quantization

Raster maps usually contain numerous colors due to the scanning or compression processes. To extract individual text layers, our supervised approach incorporates color segmentation techniques from our previous work [8] to reduce the number of colors in the maps for generating a color palette with a limited number of colors (also see [16] for a color segmentation technique that handles historical maps). Figure 3 shows an example scanned map, and Fig. 3b shows the quantized map with 16 unique colors after color segmentation.



(a) An example map area with 80,421 unique colors



(b) The quantized map with 16 unique colors after color segmentation

Fig. 3 An example map tile and the color quantization results with their *red*, *green*, and *blue* (RGB) color cubes

3.2 User labeling

Once the quantized map is generated, a user labels the quantized map to provide a text example for each text color in the quantized map. A text example is a rectangle that must be large enough to cover a string of at least two characters, and the text example does not have to cover only text pixels in the map. The user can also provide non-text examples to help the supervised technique identify the text colors. A non-text example is a rectangle that covers *only* non-text colors in the raster map.

The user-labeling interface works as follows: the user first clicks on the map and then selects the size of the example area. The user can rotate the selection area to label non-horizontal text if needed. Figure 4 shows the user interface and a text example. The text example is automatically rotated to the horizontal direction if the user selects a non-horizontal text string.

3.3 Automatically identifying text colors using text examples

In each text example, there exist one or more colors that represent text in the raster map (i.e., the text colors). To identify the text colors in a text example, we exploit the fact that the character pixels in the text example are spatially near each other and constitute a horizontal string, namely the *horizontal-string property*.

We first decompose a text example into a set of images so that every decomposed image contains only one color from the text example. For example, for the text example shown in Fig. 5a, the first row of Fig. 5 shows the four decomposed images (background is shown in black). With the decomposed images, we use the morphological operator implementation of the run-length smoothing algorithm (RLSA) to determine the decomposed images that

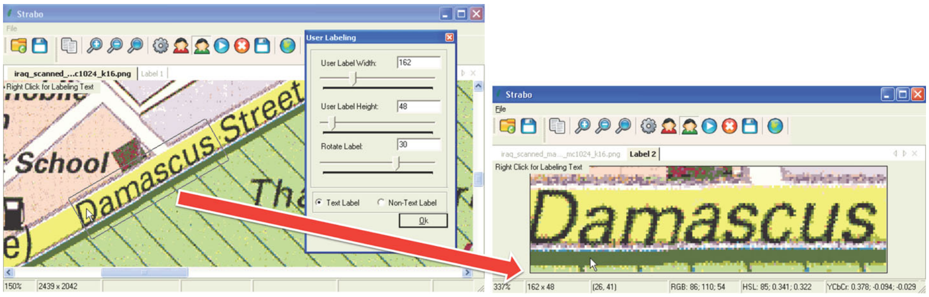
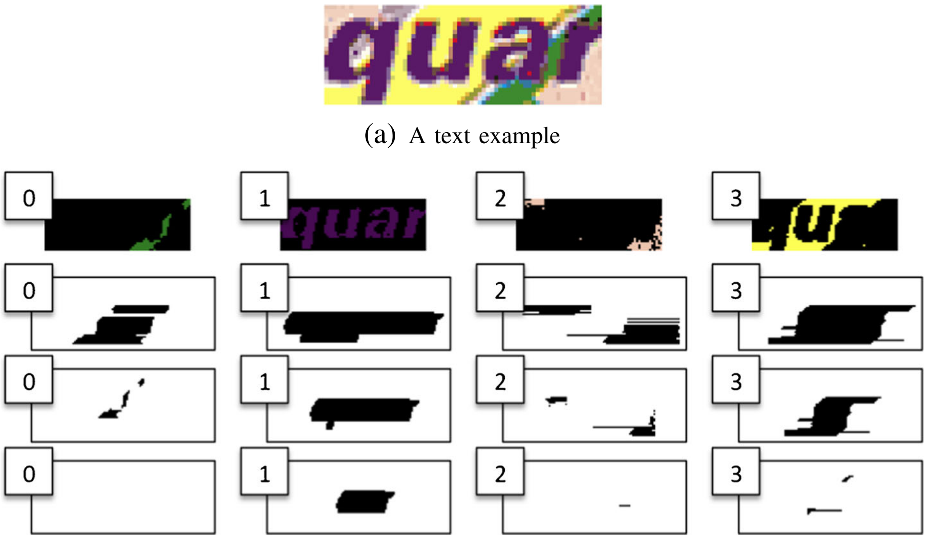


Fig. 4 The user labeling interface, and the selected text example

contain text pixels. The RLSA is the combination of the closing operator followed by the erosion operator and is commonly used in document analysis techniques to identify string blocks from character pixels [24, 31].

The closing operator is the dilation operator followed by the erosion operator. For a background pixel in a decomposed image, if there exists a foreground pixel in the horizontal direction within the distance of H, the dilation operator converts the background pixel to the foreground. Then for each foreground pixel (including the ones converted by the dilation operator), if there exists a background pixel in the horizontal direction within the distance of H, the erosion operator converts the foreground pixels to the background. Since the character pixels are horizontally near each other in the text sting, we exploit the number of the remaining foreground pixels after the RLSA as a measure to determine if the foreground pixels in a decomposed image represent text in the raster map.



(a) A text example

(b) The decomposed images and the RLSA results

Fig. 5 Identifying the text colors

Because the height of a text example, *H*, is larger than the height of any character in the text example, and the character height is usually larger than the character width, the horizontal pixel distance between two character pixels in a decomposed image should be less than *H*. Therefore, we employ the closing operator with structure elements of height equal to one pixel and width equal to *H* to expand the foreground area for connecting character pixels and to grow a string blob. The first row in Fig. 5b shows the decomposed images (background is shown in white) of Fig. 5a. The second and third rows in Fig. 5b show the results after applying the dilation and erosion operators to the decomposed images. After the closing operator, we use the erosion operator again with a structure element of height equal to one pixel and width equal to *H* again to further eliminate false-positive branches of the string blob. The fourth row in Fig. 5b shows the results after applying the erosion operator (background is shown in white). In the example in Fig. 5, *Image 1* in Fig. 5b has the most remaining foreground pixels after the RLSA, so the color of the foreground pixels in *Image 1* is identified as the text color.

There are two exceptions to using the *horizontal-string property* to identify text colors in a text example. One exception is that when background with a uniform color exists in a text example, the color of the foreground pixels in the decomposed image that represents the background can be misidentified as the text color. This is because the pixels of the decomposed image that represents the background are horizontally near each other. A second exception is when the characters in a text example are represented by multiple colors (i.e., pixelated, non-solid characters) and hence each of the text colors represents only a small portion of the foreground pixels. In both cases, the user would need to provide examples that contain non-text colors. If one or more non-text examples exist, we apply the RLSA only to the decomposed images that do not contain the colors in any of the non-text examples.

We process every text and non-text example to identify a set of text colors from each text example and use the identified text colors to extract text layers from the quantized map. For example, the left image in Fig. 6a shows a text example, "Antiqui", and the left image in Fig. 6c shows the extracted black text layer using the identified text color from Fig. 6a. The right image in Fig. 6a shows a text example, "ALHI", but the uniform background in this text example can be misidentified as the text color. In this case, the user provides the non-text example in Fig. 6b together with the text example of "ALHI" to extract the text layer of blue characters as the left image shown in Fig. 6d.

In the examples of Fig. 6c and d, there are still non-text objects in the extracted text layer. This is because the non-text objects have the same color as the text. To remove these non-text objects, we use a connected component analysis approach based on the character sizes in the text examples. We compute the average size of the characters in the text examples and then filter out the connected components in the extracted text layer that are smaller than half or larger than twice the average size (this filtering rule is determined empirically).

For a connected component, *A*, and its bounding box, *Abx*, the size of *A* is given by:

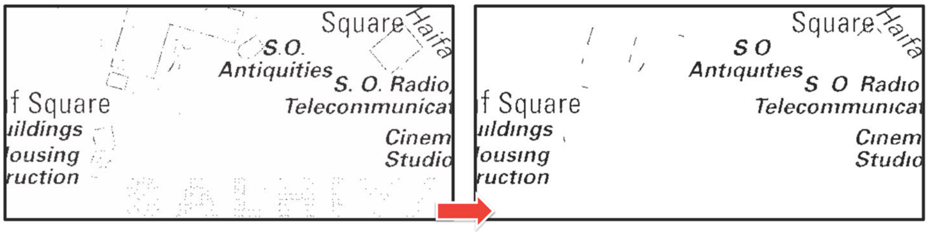
$$Size = Max(Abx.Height, Abx.Width) \tag{1}$$

Figure 6c and d show the results after this filtering step. The characters that overlap a large connected component could also be removed, such as the 'S' shown in Fig. 6d. Since most overlapping features in a map have different colors, this case is not common. Text separation algorithms [2] can be used to remove the overlapping lines and recover the overlapping characters but the recovered characters might be damaged and difficult for OCR.

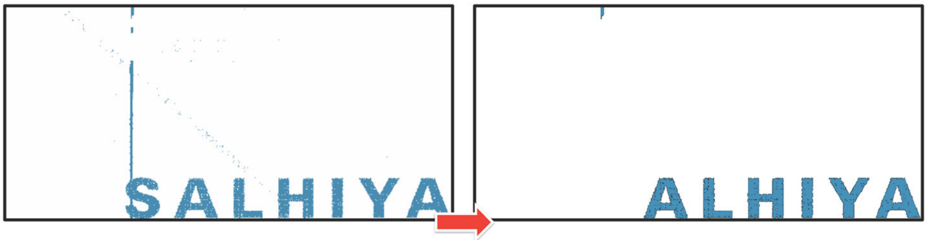
Antiqui
ALHI

(a) The text example for black and blue characters

(b) The non-text example



(c) The extracted black text layer before and after post-processing



(d) The extracted blue text layer before and after post-processing

Fig. 6 Examples of the **Text Layer Extraction** step

4 Text label recognition

The **Text Label Recognition** step is an automatic technique for recognizing text labels in the separated text layers from the **Text Layer Extraction** step. The text layers can contain multi-oriented text strings of various character sizes and curvatures. This automatic technique includes three components: (i) The conditional dilation algorithm (CDA), (ii) The single-string orientation detection algorithm (SSOD), and (iii) Optical character recognition (OCR).

The CDA divides a text layer into overlapping tiles, locates individual text strings in each tile, and merges the identified text strings. This divide-and-conquer approach enables the CDA to process large, full-sized maps. Once individual strings are located, the SSOD detects the string orientations. Finally, the strings are rotated to the horizontal direction for a conventional OCR product to recognize their characters.

4.1 Conditional dilation algorithm (CDA)

After the **Text Layer Separation** step, we have a binary image where each connected component (CC) in the foreground is a single character or a part of a character, such as the top dot of the ‘i’. To group the CCs into strings, we present the conditional dilation algorithm (CDA). Figure 7 shows the pseudo-code of the CDA.

The CDA performs multiple iterations to expand and connect the CCs and then uses the connectivity of the expanded CCs to identify individual text strings. As shown in the ConditionalDilation function in Fig. 7, before the first CDA iteration, the CDA sets every CC as expandable. Next, in an iteration, the CDA performs two scans on the input image (the *first scan* and *second scan* in the *TestConditions* sub-function). In the first scan, the CDA tests a set of expansion conditions on *every background pixel* and *the background pixel’s foreground neighbors* to determine if a background pixel is an expansion candidate. An expansion candidate is a background pixel that can convert to the foreground for expanding a CC given the expansion conditions. Once the CDA identifies the expansion candidates, in the second scan, the CDA evaluates each *expansion candidate* to identify the pixels to convert to the foreground. If an *expansion candidate* and *the candidate’s foreground neighbors*

```
// The number of processed iterations of the conditional dilation algorithm
   IterationCounter = 0;
// The number of expandable connected components
   Expandable_CC_Counter;

MainFunction
void ConditionalDilation (int[,] image, double max_size_ratio,  double max_curvature_ratio,
double max_distance_ratio)
  FOR EACH connected component CC in image
       CC.expandable = TRUE;
  DO{ TestConditions(image, max_size_ratio, max_curvature_ratio);
       CountExpandableCC(image, max_distance_ratio);
       IterationCounter = IterationCounter +1;
  } WHILE(Expandiable_CC_Counter > 0)
EndMainFunction

SubFunction
void TestConditions (int[,] image, double max_size_ratio, double max_curvature_ratio)
// first scan
  FOR EACH background pixel BG in image
    IF( PassConnectivityTest(BG) && PassSizeTest(BG, max_size_ratio)
       && PassExpandabilityTest(BG)
       && PassStringCurvatureTest(BG, max_curvature_ratio) )
       Set BG to ExpansionCandidate;
// second scan
  FOR EACH expansion candidate EC in image
    IF( PassConnectivityTest(EC) && PassSizeTest(EC, max_size_ratio)
       && PassExpandabilityTest(EC)
       && PassStringCurvatureTest(EC, max_curvature_ratio) )
       Set EC to Foreground;
EndSubFunction

SubFunction
void CountExpandableCC  (int[,] image, double max_distance_ratio)
  Expandable_CC_Counter = 0;
  FOR EACH connected component CC in image
    IF( HasConnectedToTwoCCs(CC) ||
       IterationCounter > max_distance_ratio* CC.size)
      CC.expandable = FALSE;
    ELSE
      Expandable_CC_Counter  = Expandable_CC_Counter +1;
EndSubFunction
```

Fig. 7 The pseudo-code for the conditional dilation algorithm (CDA)

and expansion-candidate neighbors do not violate the expansion conditions, the CDA converts the expansion candidate to the foreground. After an iteration, the CDA evaluates each expanded CC (the *CountExpandableCC* sub-function) to determine whether the CC can be further expanded in the next iteration and stops when there is no expandable CC.

4.1.1 The first scan

In the first scan, the CDA checks every background pixel in the input image. If a background pixel meets all of the following expansion conditions, the CDA sets the background pixel as an expansion candidate. These conditions do not have to be checked in a fixed order.

Character connectivity condition An expansion candidate needs to connect to at least one and at most two characters. This is because the maximum neighboring characters that any character in a text string can have is two.

Character size condition If an expansion candidate connects to two characters, the sizes of the two characters must be similar. For a character, *A*, and its bounding box, *Abx*, the size of *A* is defined as:

$$Size = Max(Abx.Height, Abx.Width) \tag{2}$$

For the characters connected by expansion candidates, the size ratio between the characters must be smaller than a predefined parameter (the *max_size_ratio parameter*). For two characters, *A* and *B*, their bounding boxes are *Abx* and *Bbx*, their size ratio is defined as:

$$SizeRatio = \frac{Max(Size(A), Size(B))}{Min(Size(A), Size(B))} \tag{3}$$

This character size condition guarantees that every character in an identified text string has a similar size. We use the size ratio equal to two because some letters, such as the English letter ‘l’ and ‘e’, do not necessarily have the exact same size, even when the same font is used.

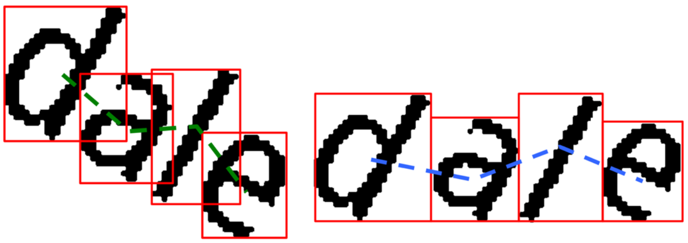
Character expandability condition An expansion candidate needs to connect to at least one expandable CC and the expandability of a CC is determined as follows: before the first CDA iteration, every CC is expandable. After each iteration, the CDA checks the connectivity of each expanded CC and if the expanded CC has already connected to two other CCs, the CC is not expandable.

Next, for the remaining expanded CCs (i.e., the ones with connectivity less than two), the CDA determines the expandability of each CC by comparing the number of iterations that have been done and the original size of each CC before any expansion. This is to control the longest distance between any two characters that the CDA can connect so that the characters in two separated strings will not be connected. For example, in our experiments, we empirically set the longest distance between two characters to 1/5 of the character size (the *max_distance_ratio* parameter). As a result, for a character of size equal to 20 pixels, the character will not be expandable after four iterations, which means this character can only find a connecting neighbor within the distance of 4 pixels plus 1/5 of the size of a neighboring CC.

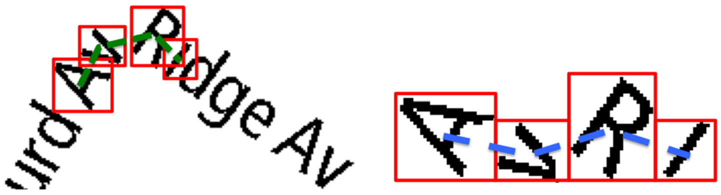
String curvature condition If an expansion candidate connects two CCs and at least one of the two CCs has a connected neighbor (i.e., together as a string with at least three characters), the curvature of the set of CCs should be less than the maximum desired curvature.

This condition allows the CDA to identify curved strings and guarantees that the characters of the text strings in different orientations will not be connected. However, determining the string curvature without knowing how the characters are aligned is unreliable. For example, considering the text string “Wellington”, if we link the mass centers or bounding-box centers of each character to represent the string curvature, the line segments linking any two neighboring characters can have very different orientations since the characters have various heights, such as the links between “We” and the one between “el”.

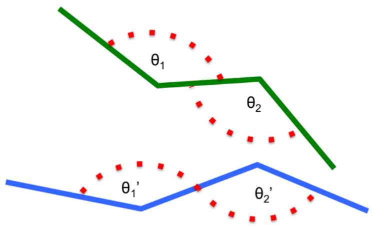
To accurately estimate the curvature of a string, the CDA first establishes a curvature baseline for the string. For example, the left image in Fig. 8a shows an example string, and the right image shows the rearranged string as if the example string is straight and in the horizontal direction. The CDA generates the rearranged string by first aligning each of the characters vertically and rearranging the characters’ positions in the horizontal direction so that the characters are not overlapped. The dashed line in the right image shows the curvature baseline of “dale”. This curvature baseline contains two connecting angles: one between “dal” and one between “ale”.

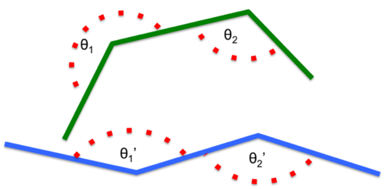


(a) The original string (left) and curvature baseline (right) of “dale”



(b) The original string (left) and curvature baseline (right) of “AvRi”





(c) θ_1 / θ_2 is similar to θ_1' / θ_2' (d) θ_1 is very different from θ_1'

Fig. 8 Testing the string curvature condition

With the curvature baseline, the CDA determines the string curvature by comparing the connecting angles in the original string to the ones in the curvature baseline. For example, Fig. 8c shows that θ_1 is similar to θ_1' and θ_2 is similar to θ_2' and hence the CDA considers the string "dale" as a straight string (i.e., every original connecting angle is similar to its corresponding one). Figure 8d shows an example where θ_1 is very different from θ_1' and hence the CDA considers the string "AvRi" as a curved string.

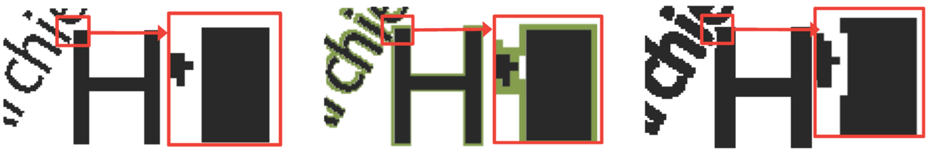
The CDA uses a curvature parameter to control the maximum desired curvature of a text string (the *max_curvature_ratio* parameter). If the difference between one connecting angle of a string and the corresponding angle in the string's curvature baseline is larger than the curvature parameter, the string violates the string curvature condition. For example, with the curvature parameter set to 30 % from the curvature baseline, any string with curvature within 138° (180° divided by 130 %), to 234° (180° multiplied by 130 %) will be preserved.

4.1.2 The second scan

The second scan checks each expansion candidate using the same conditions in the first scan. During the first scan, the CDA does not have the knowledge of the locations of every expansion candidate before the first scan ends. Therefore, two connecting expansion candidates could violate the expansion rules. For example, Fig. 9a shows two characters, 'H' and 'i', that should not be connected because of the size limitation. Fig. 9b shows the results after the first scan, where the green pixels are the identified expansion candidates. After the first scan, the CDA marks all background pixels that directly connect to the two characters as expansion candidates. If the distance between the two characters is two pixels, such as the areas in the red rectangle shown in Fig. 9a, after the first scan, the CDA fills up the two-pixel area with expansion candidates and the two characters are then connected. Therefore, we need the second scan to verify the expansion candidates. Figure 9c shows the results after the second scan.

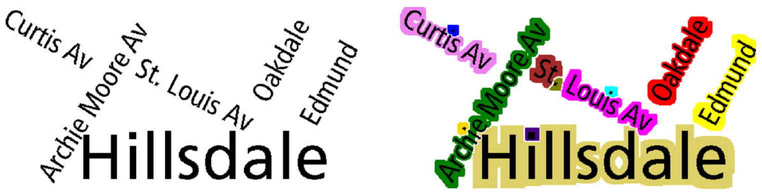
4.1.3 The CDA output

After the CDA stops when there are no expansion candidates, each connected component of the expansion results is an identified text string. For example, in Fig. 10, the set of color blobs are the expansion results (each color represents a connected component), and the black pixels overlapped with a color blob belong to an identified string. In Fig. 10, the CDA does not group small CCs correctly, such as the dot on top of the character 'i' . This is because these small CCs violate the character size condition. The OCR system will recover these missing small parts in the character recognition step, which is more robust than adopting special rules for handling small CCs in the CDA.



(a) An example map (b) After the first scan (c) After the second scan

Fig. 9 Using the second scan to determine actual pixel for expansion (expansion candidates are shown in *green* and background is shown in *white*)



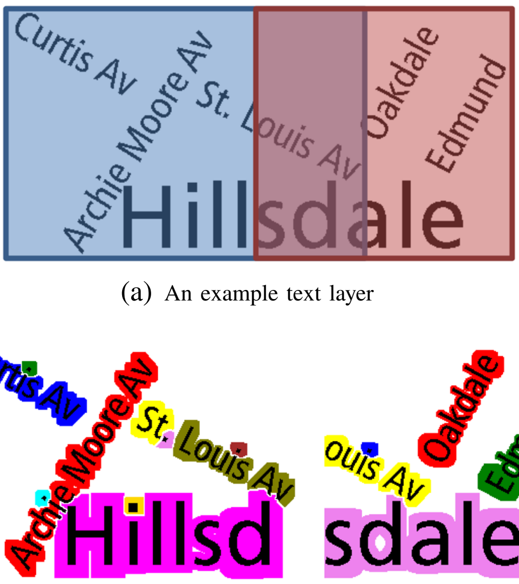
(a) An example layer (b) The CDA result

Fig. 10 The CDA output

4.1.4 The CDA's divide-and-conquer approach

Because scanned maps are usually large images (a typical 350 dot-per-inch (DPI) scanned map can be larger than 6000x6000 pixels), loading the entire map into memory for the CDA to process is very often impractical and sometimes impossible. Therefore, the CDA divides a raster map into overlapping tiles and processes each tile to identify individual text labels. Figure 11a shows an example text layer with two overlapping tiles. After the CDA processes all the tiles, the algorithm merges the identified text strings from each tile as one set of text strings for the entire raster map.

Before processing each tile, the CDA first removes the connected components that touch each tile's borders since the touching connected components might only be a portion of a character. The overlapping area should be larger than any of the characters in the text layer so that the characters near the tile borders always exist in one of the tiles. For example, Fig. 11b and c shows the text identification results of the two overlapping tiles using the CDA. In the left tile, the character 'a' of the text string "Hillsdale" in the text layer is on



(a) An example text layer

(b) The left tile (c) The right tile

Fig. 11 The divide-and-conquer processing

the tile border and hence the CDA removes the ‘a’ before applying the conditional dilation algorithm. For the identified strings in two neighboring tiles, the CDA merges the strings that contain one or more characters that are the same. For example, Fig. 11b and c show that the conditional dilation algorithm identifies the text string “Hillsd” from the left tile and the text string “sdale” from the right tile. Since the two characters “sd” in the original text layer exist in both text strings, the CDA merges the two strings into one as “Hillsdale”.

4.2 Single-string orientation detection algorithm (SSOD)

Skew correction is well developed in modern OCR techniques to detect the orientation of document images; however, classic skew correction can only apply to multi-line and multi-word documents since the line spacing and the word spacing is exploited to detect the tilt angle, such as the morphological-operator-based RLSA method [24].

To detect the orientation of a single string, we present a single-string orientation detection algorithm (SSOD) based on the morphological-operator-based RLSA [6]. The SSOD employs morphological operators (i.e., the closing and erosion operators) with dynamically generated structure elements. Figure 12 shows the pseudo-code of the SSOD.

Given a string, the SSOD first rotates the string image from 0° to 179°. Then the SSOD uses the closing operator with a structure-element wider than the character spacing to merge nearby characters in the horizontal direction. Since the character size is generally larger than the character spacing, we use the average size of the connected components in the string as the width of the structure element of the closing operator. This average size is called $AvgSize$ in the pseudo-code. For each rotated string, the SSOD applies the closing operator using a structure element of height equal to one and width equal to $AvgSize$ to grow the string blobs, as the examples shown in the middle row in Fig. 13.

After the closing operator, if a string is in the horizontal direction, there exist character pixels that have no neighboring background pixels along the horizontal direction within

```
// The list for storing the rotated string images
RotatedImageList;

Function int SSOD (Image string_image)
    AvgSize = FindAvgCCSize(string_image);
    For angle = MinRotation to MaxRotation {
        rotated_image = Rotate(string_image, angle);
        RotatedImageList.Add(rotated_image);
    }
    MaxWidth = FindMaxStringWidth();
    MaxForegroundPixelCount =
                            RLSA(AvgSize , MaxWidth);
    For each rotated image RI in RotatedImageList {
        if(RI. ForegroundPixelCount ==
                                MaxForegroundPixelCount )
            return RI.RotatedAngle;
    }
```

```
Function double FindAvgCCSize(Image string_image)
    double AvgSize;
    For each connected component CC in string_image {
        AvgSize = AvgSize + CC.Size;
    }
    return AvgSize/string_image .TotalNumberOfCCs;

Function int FindMaxStringWidth()
    StringWidthList;
    For each rotated image RI in RotatedImageList {
        Pixel left =
            FindTheLeftMostForegroundPixel(RI);
        Pixel right =
            FindTheRightMostForegroundPixel(RI);
        StringWidthList.Add(right.X – left.X);
    }
    return StringWidthList.MaxValue;

Function int RLSA(AvgSize , MaxWidth )
    RemainingForgroundPixelList;
    For each rotated image RI in RotatedImageList {
        RI = Closing(RI, AvgSize, MaxWidth);
        RI = Erosion(RI, MaxWidth);
        int FGC = RI.ForegroundPixelCount;
        RemainingForgroundPixelList.Add(FGC);
    }
return RemainingForgroundPixelList.MaxValue;
```

Fig. 12 The pseudo-code for the single-string orientation detection algorithm

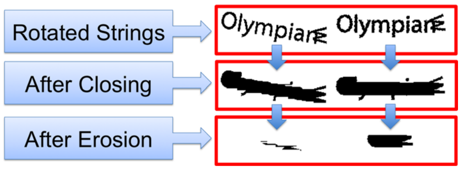


Fig. 13 Detecting string orientation using the morphological-operators-based RLSA

a distance similar to the string length. The SSOD utilizes the erosion operator to identify these character pixels by erasing the character pixels that have one or more neighboring background pixels along the horizontal direction within a distance threshold. This distance threshold is the width of the erosion structure element and is determined using the string length.

Since we do not know the actual length of the string, we first compute the maximum string length using the longest horizontal length of the rotated strings (the *FindMaxStringWidth* function in the pseudo-code). This longest horizontal length is called $MaxWidth$ in the pseudo-code. Directly using the $MaxWidth$ as the structure element width for the erosion operator might erase every rotated image if the string has characters of very different sizes in height or the string is curved (i.e., every character pixel has one or more neighboring background pixels along the horizontal direction within the $MaxWidth$). Considering that a string has the longest horizontal length when the string is placed close to the horizontal direction (the string orientation is near 0°) and has the shortest horizontal length when the string is placed close to the vertical direction (the string orientation is near 90°), we use the horizontal length as if the string is rotated 45° as the width of the erosion structure element to prevent over-erosion. As a result, the SSOD uses the erosion operator with a structure element of height equal to one and width equal to the $MaxWidth$ multiplied by cos(45°) to shrink the area of the merged characters.

The bottom row in Fig. 13 shows example results after applying the erosion operator where the horizontal string has more remaining pixels than the tilted string. The SSOD then identifies the actual horizontal string among the rotated strings using the number of remaining pixels after applying the erosion operator. We do not use the rotated string that has the $MaxWidth$ as the actual horizontal string. This is because the characters in a string can have various shapes and the rotated string that has the $MaxWidth$ can be a few degrees off from the horizontal direction.

The SSOD only applies on the strings having more than three connected components. This is because the detected orientation of a short string can be dominated by the character height using the RLSA. Since short strings in a raster map are usually part of a longer string, we search from the centroid of a short string for nearby strings and use the orientations of the nearby strings as the short string's possible orientations. For example, the most common short strings in our test maps are "Av" as avenue and "Pl" as place, which are all part of the road names. We dynamically generate a distance-threshold based on the size of the bounding box of each short string to limit the search space.

4.3 Optical character recognition

Once the SSOD identifies the string orientations, we first rotate each string clockwise and counterclockwise to the horizontal direction according to its possible orientations (short strings might have more than one detected orientation depending on the number of its neighboring strings) to generate a set of rotated strings. Then we send all rotated strings to a commercial OCR product called ABBYY FineReader 10.

The ABBYY FineReader 10 is a standard OCR product that supports text recognition from a variety of images, including scanned documents and photos (i.e., text with a background image), and has built-in font types and dictionaries for 186 languages (including English). For an input image, the ABBYY FineReader automatically identifies areas that contain text strings and then automatically recognizes the character of the text strings. Because of the various sets of built-in font types, training is not required for the character recognition process. In the ABBYY FineReader results, each recognized character is labeled as either *confident* or *suspicious*. The *suspicious* label means that the ABBYY FineReader does not have enough evidence to determine that the recognition result is correct and further manual verification is required.

For each of rotated strings, we calculate a recognition confidence and select the rotated string with the highest returned recognition confidence as the correctly oriented horizontal string (i.e., not the upside-down one). The recognition confidence is calculated using the number of connected components in the string, the number of recognized characters, and the number of suspicious characters to calculate the recognition confidence. Formally, *NRC* is the number of recognized characters, *NSC* is the number of suspicious recognized characters, and *NCC* is the number of connected components of a text string, the recognition confidence is given by:

$$RecognitionConfidence = \frac{NRC - NSC}{NCC} \tag{4}$$

The *NCC* is generated by the CDA, and the *NRC* and *NSC* are from the ABBYY FineReader. Our approach does not rely on specific OCR functions/results of the ABBYY FineReader. The *NRC* and *NSC* can be found in most standard OCR products.

If two rotated strings have the same recognition confidence, we show both recognition results in the final results. For short strings with fewer than three characters, if the recognition confidence is less than 50 %, we discard the results since the strings are likely to be non-text objects. For longer strings, if the recognition confidence is less than 50 %, we set the number of suspicious characters to zero and then recalculate the recognition confidence. This is because it is very likely that the quality of the original map is poor so the OCR software marks most of the recognized characters as suspicious.

5 Experiments

We have implemented the techniques described in this paper in our map processing system called Strabo. To evaluate our technique, we tested Strabo on 15 maps from 10 sources, including 3 scanned maps and 12 computer-generated maps (directly generated from vector data).[1] These maps contain non-homogeneous text of numeric characters and the English alphabet. Table 1 shows the information of the test maps and their abbreviations used in this section. We manually identify the characters and words in these test maps as the experiment ground truth.

Figure 14 shows the example areas and text of each test map. The scanned maps show poor image quality compared to the computer-generated maps. In addition to the image quality, as shown in Fig. 14f, the text labels in the computer-generated maps of Google,

[1]The information for obtaining the test maps can be found on: http://www.isi.edu/integration/data/maps/prj_map_extract_data.html

Table 1 Test maps for the experiment

Map source (abbr.)	Map type	# Char/Word
International Travel Maps (ITM)	Scanned	1358/242
Gecko maps (GECKO)	Scanned	874/153
Gizi map (GIZI)	Scanned	831/165
Rand McNally (RM)	Computer generated	1154/266
UN Afghanistan (UNAfg)	Computer generated	1607/309
Google maps (Google)	Computer generated	401/106
Live maps (Live)	Computer generated	233/64
OpenStreetMap (OSM)	Computer generated	162/42
MapQuest maps (MapQuest)	Computer generated	238/62
Yahoo maps (Yahoo)	Computer generated	214/54

Live, OSM, MapQuest, and Yahoo contain pixilated non-solid characters, which is especially difficult for an OCR system to recognize even if the text labels are in the horizontal direction.

During the **Text Layer Extraction** step, Strabo generated a set of quantized map for these scanned maps. The user manually selected the quantized map that had the clearest text appearance to provide examples of text and non-text areas. The optimal number of the quantized colors should be close to the number of representative colors used in the map (more details are in our previous work [8]). Strabo lets user decide the optimal number based on the visual appearance of each input map. For example, the user might choose to generate a set of quantized map with the numbers of quantized colors ranging from 4 to 16 on maps with a few representative colors, and could use the range from 32 to 512 on complex maps.

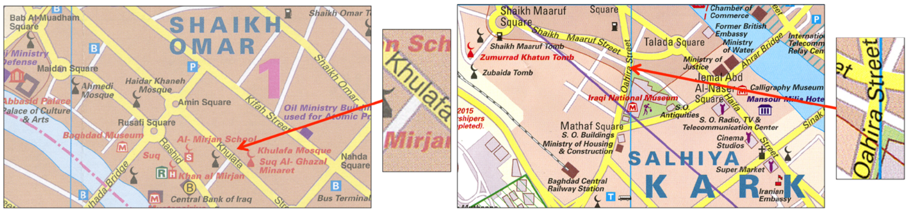
We utilized Strabo together with the ABBYY FineReader 10 to recognize the text labels in the test maps. Strabo sent the FineReader one string each time and each string had white background and black foreground. We specifically designed this setting to reduce the content analysis work that the FineReader had to perform to have an objective comparison since the details of the core algorithms of the FineReader are not available (especially the content analysis algorithm).

For comparison, the FineReader was also tested alone without Strabo. We chose to test the FineReader on the original test maps without any additional process from Strabo since the FineReader is a self-contained OCR system that has built-in color segmentation, noise removal, and zoning capabilities (that cannot be turned off individually) and is designed to recognize text from complex background (e.g., text on photos).

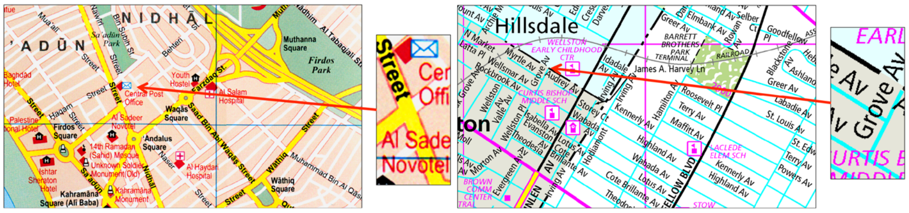
5.1 Experimental results

In this section, we report the recognition accuracy at the character and word level to evaluate the overall Strabo performance. We report the number of user interaction steps (labels) to measure the required user effort for the overall text recognition process in Strabo and the performance of the **Text Layer Extraction** step. This is because the goal of the **Text Layer Extraction** step is to help reduce the required user interaction during the text recognition process.

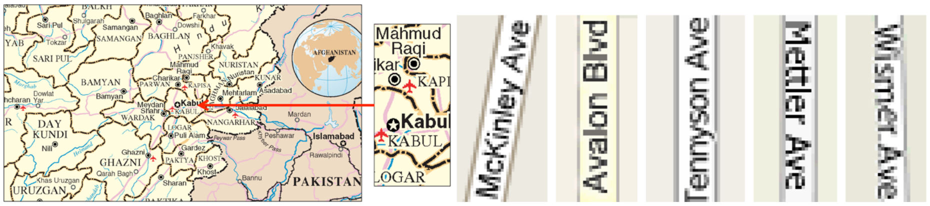
Although the **Text Layer Extraction** step itself may be seen as a binary classification problem (i.e., classifying map pixels into text and non-text groups), using a precision-recall



(a) ITM map (b) GECKO map



(c) GIZI map (d) RM map



(e) UNAfg map (f) From left to right, Google, Live, OSM, MapQuest, and Yahoo maps

Fig. 14 Example test maps

metric to measure the pixel classification result does not provide a good indicator of the final text recognition rate. For example, missing the dot in ‘i’ does not usually affect the recognition result, but missing the upper part of ‘b’ can lead to an incorrect recognition result (i.e., the character would be identified as ‘D’). In addition, the classification ground truth cannot be objectively defined. For example, a pixelated character can be recognized with or without all the “shadow” pixels surrounding the character. The goal of this step is to extract all text colors (not text pixels) from the map. There could be pixels of non-text objects that were in the same color of text, and these non-text objects will be identified and removed in the later steps (of which the performance were measured using the overall text recognition precision/recall).

Table 2 shows the number of extracted text layers, text colors (in the quantized maps), and text and non-text examples used for extracting the text layers. The nature of the original map (e.g., image quality, scanned vs. computer generated) and the color segmentation process in the **Text Layer Extraction** step dictated the number of text and non-text examples needed for each input map. For the ITM, GECKO, and GIZI maps, the color segmentation process generated solid characters. The RM and UNAfg maps did not require the color segmentation process because they contain only a few colors and have solid characters in the original map. The ITM, GECKO, and GIZI maps after the color segmentation process contain solid characters. Therefore, fewer examples were needed for extracting a text layer from the RM, UNAfg, ITM, GECKO, and GIZI maps. For the other maps that have pixilated, non-solid

Table 2 The number of extracted text layers, text colors (in the quantized maps), and the number of user labels for extracting the text layers

Source	# of Text Layers	# of Text Colors	# of Text/Non-Text Examples
ITM	3	7	6/3
GECKO	3	3	3/2
GIZI	2	2	2/2
RM	2	2	2/2
UNAfg	2	2	2/2
Google	2	45	2/4
Live	2	41	6/12
OSM	2	200	3/10
MapQuest	2	75	3/6
Yahoo	2	31	1/1

characters, the higher numbers of examples were used due to the fact that more non-text examples were used. Traditionally (without Strabo), when a user needs to manually select one (or more) pixels for each of the text colors, the number of user interaction steps is a function of the number of colors in the text layer. In contrast, with Strabo the user provides examples that cover a number of pixels with multiple colors so that even if the number of text colors increased significantly, the number of user interaction steps would still be low. For example, the OSM contained 200 text colors but the number of text/non-text examples was 3/10 compared to 2/2 for GIZI maps where the number of text colors was 2.

Table 3 shows the numeric results from using Strabo and using the FineReader itself to recognize text strings in the 15 test maps. We empirically set the size ratio (the *max_size_ratio* in the CDA pseudo-code) to 2, the distance ratio (the *max_distance_ratio* in the CDA pseudo-code) to 1/5, and the desired curvature ratio (the *max_curvature_ratio* in the CDA pseudo-code) to 30 %.

A larger value of the size ratio will cause the CDA to link characters with different font sizes, and a smaller values can lead to broken strings (depending on the font type). The distance ratio determines the maximum character spacing within a string. In a map with crowded strings that are in the same orientation, the distance ratio should be set to a lower number to prevent the incorrect merging of individual strings. Further effort is required to automatically determine the best size ratio and distance ratio for individual maps.

The desired curvature ratio controls the breaking point(s) of a curved string. A lower number for the curvature ratio will make the CDA break the strings that are slightly curved, which can be a desired outcome if the OCR process can only handle strictly straight text strings. The FineReader does handle slightly curved strings and hence we used the desired curvature ratio of 30 %.

Strabo extracted 6,708 characters and 1,383 words from the test maps and ABBYY FineReader 10 extracted 2,956 characters and 655 words. Strabo produced higher numbers compared to only using FineReader in all metrics, especially the recall. This is because Strabo successfully analyzed the map contents and grouped the multi-oriented and multi-sized characters into individual text strings for OCR. Moreover, Strabo correctly identified curved strings that have their curvature within the desired curvature ratio (30 %), such as the example shown in Fig. 16a.

Table 3 Text recognition results using Strabo and ABBYY (F. is the F-Measure)

Source	System	Char Precision	Char Recall	Char F.	Word Precision	Word Recall	Word F.
ITM	Strabo	93.6 %	93.3 %	93.5 %	83.3 %	82.6 %	82.9 %
	ABBYY	86.4 %	45.6 %	59.7 %	57.5 %	33 %	41.9 %
GECKO	Strabo	93.4 %	86.3 %	89.7 %	83.1 %	77.1 %	80 %
	ABBYY	77.8 %	41 %	53.7 %	66.2 %	37.2 %	47.7 %
GIZI	Strabo	95.1 %	77.3 %	85.3 %	82 %	63.6 %	71.6 %
	ABBYY	71.3 %	16. %	26.7 %	51.4 %	10.9 %	18 %
RM	Strabo	93.4 %	94 %	94.1 %	87.9 %	84.9 %	86.4 %
	ABBYY	71.8 %	10.4 %	18.1 %	23.5 %	3 %	5.3 %
UNAfg	Strabo	91.5 %	88 %	89.7 %	82.3 %	80.2 %	81.3 %
	ABBYY	65.6 %	56 %	60.4 %	34.8 %	36.5 %	35.7 %
Google	Strabo	97.3 %	91.7 %	94.4 %	89.2 %	85.8 %	87.5 %
	ABBYY	0 %	0 %	0 %	0 %	0 %	0 %
Live	Strabo	94.7 %	93.5 %	94.1 %	75.3 %	76.5 %	75.9 %
	ABBYY	51.8 %	47.6 %	49.6 %	47.8 %	53.1 %	50.3 %
OSM	Strabo	95.4 %	77.7 %	85.7 %	74.3 %	69 %	71.6 %
	ABBYY	0 %	0 %	0 %	0 %	0 %	0 %
MapQuest	Strabo	91.3 %	84 %	87.5 %	81 %	75.8 %	78.3 %
	ABBYY	0 %	0 %	0 %	0 %	0 %	0 %
Yahoo	Strabo	69.7 %	63.5 %	66.5 %	43.1 %	40.7 %	41.9 %
	ABBYY	0 %	0 %	0 %	0 %	0 %	0 %
Avg.	Strabo	92.7 %	87.9 %	90.3 %	82 %	77.5 %	79.7 %
Avg.	ABBYY	71.9 %	30 %	42.4 %	46.1 %	20.6 %	28.5 %

The FineReader did not do well on identifying text regions from these test maps because of the non-homogenous text content in the maps. In particular, ABBYY FineReader 10 could not detect any text region from the Google, OSM, MapQuest, and Yahoo maps, and hence the precision and recall are 0 at both the character and word levels. Most of the correctly recognized strings were either in or slightly skewed from the horizontal or vertical directions.

To obtain the best text recognition results that the FineReader could achieve (with additional manual work), we rotated each of the test maps from Google, Live, OSM, MapQuest, and Yahoo maps (a total of 10 maps) from 0° to 355° using a 5° increment (a total of 72 images for each map) so that every string was horizontally placed in at least one of the 72 images. Then we used the FineReader to process these rotated images and manually selected the correctly recognized characters/words from the recognition results of all 72 images.

If a character/word was correctly recognized in one (or more) of the 72 images, we counted it as a correctly recognized character/word. For example, given a word "Main" in a test map, after we processed the 72 images using the FineReader, if one or more of the recognition results from the 72 images contained the correctly recognized word "Main", we recorded one correctly recognized word and four correctly recognized characters. The recognition precision of this manual process cannot be objectively calculated. This is because the final results were manually selected from all of the results, and the total number of recognized character/words (the denominator for calculating precision) might change if we changed the angle of rotation. For example, a 2° increment to rotate the original map would generate 180 test images. Since a character would be correctly recognized in only a few of these test images, increasing the number of the test images would increase the denominator for calculating precision and thus reduce the precision.

Table 4 shows the recognition results from Strabo, the FineReader with a map rotated a 5° increments, and FineReader on the original map. For the Live maps, the multiple rotations helped to improve the recall since some of the non-horizontal strings were correctly identified from one of the rotated images. For other map sources, the additional rotations did not help much. In most cases, the FineReader could not identify any text regions even when the strings were rotated to the horizontal direction.

5.2 Result analysis

Overall Strabo achieved accurate text recognition results on both the character and word levels. The errors in Strabo's results came from several factors:

(i) The poor image quality of the test maps could result in poor quality of the text layers, such as broken characters or the existence of non-text objects in the text layer. In our experiments, the GIZI map had the worst image quality among the scanned maps, and hence the result numbers of the GIZI map were the lowest among the maps with solid characters.

(ii) The similarity between symbols led to false positives. There were many short strings of "Pl" for place in our test maps, and most of them were misidentified as "PI" (a capital 'p' and a capital 'i'). This is because in some font types, the capital 'i' is

Table 4 Text recognition results using Strabo, ABBYY with manual image rotation and result selection, and ABBYY alone

Source	System	Char Recall	Word Recall
Google	Strabo	91.7 %	85.8 %
	ABBYY with rotated maps	3 %	2.8 %
	ABBYY	0 %	0 %
Live	Strabo	93.5 %	76.5 %
	ABBYY with rotated maps	75.9 %	73.4 %
	ABBYY	47.6 %	53.1 %
OSM	Strabo	77.7 %	69 %
	ABBYY with rotated maps	0 %	0 %
	ABBYY	0 %	0 %
MapQuest	Strabo	84 %	75.8 %
	ABBYY with rotated maps	4.6 %	6.5 %
	ABBYY	0 %	0 %
Yahoo	Strabo	63.5 %	40.7 %
	ABBYY with rotated maps	0 %	0 %
	ABBYY	0 %	0 %

printed as ‘l’ and the OCR software mismatched the two letters. Moreover, a string could be misidentified as a totally different string when the string was upside down, especially short strings, such as “Pl” and “ld” or “99” and “66”. Figure 15 shows an example of the string “Zubaida” could be misidentified as “epieqnz”. This type of false positive resulting from similar symbols is very difficult to remove if the actual orientation of the string is unknown. One possible solution is to introduce additional knowledge of the map area to filter out unlikely results, such as “ld” and “PI” (a capital ‘p’ and a capital ‘i’).

(iii) Strabo could not detect correct orientations for significantly curved text strings, and the OCR software could not recognize all characters in curved strings. Figure 16a shows two examples of curved strings detected by Strabo and the rotated horizontal strings according to their detected orientations. If the string was slightly curved, such as the first row in Fig. 16a, Strabo could detect correct orientation so that one of the rotated horizontal strings is correctly oriented (the third string on the first row in Fig. 16a). For the significantly curved strings, such as the second row in Fig. 16a, Strabo could not detect the correct orientation. However, even if Strabo identified the correct orientation for the slightly curved strings, the OCR software could not recognize all characters of the string because the string is not straight. To overcome this problem, Strabo could use a lower threshold on comparing the connecting angle to the baseline for breaking any curved strings. Then, post-processing to merge the recognition results of the pieces of the curved strings can be used to recover the broken strings.

(iv) The CDA might not group strings with wide character spacing. The characters in a string that had wide character spacing were not correctly grouped since the CDA used a distance threshold depending on the sizes of the characters. For example, Fig. 16b the string “Hindu Kush” in the UNAfg map were not identified correctly.

(v) The OCR software could not recognize some of the pixilated, non-solid characters. For the pixilated non-solid characters, a character is not necessarily an individual connected component, and the CDA might generate incorrect string blobs. The Yahoo maps had the most pixilated characters and hence the result numbers were the lowest. In addition to the incorrect string blobs, the pixilated characters are difficult for a machine to recognize, although humans can recognize the pixilated characters from a distance.

(vi) The CDA might group characters with non-text objects. If there exist non-text objects in the CDA input and a non-text object was close to one end of a string and has a similar size as the ending character, the CDA would connect the end character to the non-text object. This would result in incorrectly detected orientation. A connected-component filter can be used to post-process the extracted text pixel for removing this type of error. However, the connected-component filter would need careful parameter settings and might also remove characters.

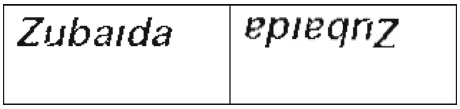
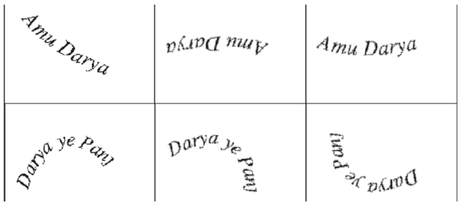
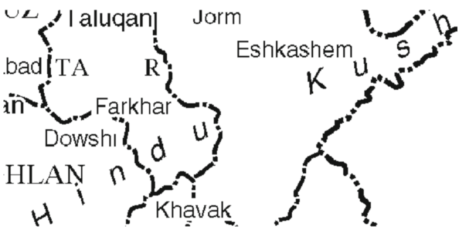


Fig. 15 The string “Zubaida” can be mis-identified as “epieqnz”

Fig. 16 Examples of the strings that could lower the recognition accuracy



(a) Curved strings



(b) Wide character spacing

(vii) Since the FineReader is a desktop version OCR software, the recognition results included only limited information. We could incorporate an OCR software development kit (SDK), such as Tesseract, to obtain more detailed information of the recognition results, such as the confidence level for each character, to improve the overall accuracy.

5.3 Computation time

We built Strabo using Microsoft Visual Studio 2008 running on a Microsoft Windows 2003 Server powered by a 3.2 GHz Intel Pentium 4 CPU with 4GB RAM. The average processing time for the CDA on a 1688x1804-pixels text layer of 626 characters was 37 seconds, for a 2905x2384-pixels text layer of 1092 characters was 39 seconds, and for a 850x550-pixels text layer of 78 characters was 2.6 seconds. Dominant factors of the computation time are the image size, the number of characters, and the shortest distance between two characters in a string (a longer distance requires more iterations for the CDA to converge). The average processing time for detecting the orientation on a string longer than three characters was 2.2 seconds (on a total of 922 strings), and the dominant factor on the computation time was the length of a string.

6 Conclusion and future work

We presented a general approach that requires only a few user interaction steps for text recognition from raster maps. We compared our approach to a state-of-art commercial OCR product using 15 raster maps from 10 sources. We showed that our approach enabled the commercial OCR product to handle raster maps and together produced significantly higher text recognition accuracy than using the commercial OCR alone. We demonstrated that our approach can be easily integrated with a commercial OCR product to support text recognition from documents for which classic layout analysis techniques do not work.

In the future, we plan to use multiple maps of the same region to improve the OCR accuracy. For example, for a set of maps covering the same region, we can first recognize text labels in each of the maps and use the mutual information to help improve the overall recognition accuracy. We can also build gazetteers from historical maps and compare the built gazetteers with data from other sources for spatial-change analysis. We plan to broaden the coverage of our technique to handle documents with touching characters, such as by incorporating a character segmentation method [29]. We will explore the options to incorporate image segmentation techniques such as Lazy Snapping [18] and GrabCut [27] in the step to extract text layers. Lazy Snapping [18] and GrabCut [27] can help identify the text boundaries and hence further reduce the amount of manual work and user decisions (such as selecting a quantized map for text recognition).

Acknowledgment This research is based upon work supported in part by the University of Southern California under the Viterbi School of Engineering Doctoral Fellowship.

References

1. Adam S, Ogier J, Cariou C, Mullot R, Labiche J, Gardes J (2000) Symbol and character recognition: application to engineering drawings. Int J Doc Anal Recog 3(2):89–101
2. Cao R, Tan CL (2002) Text/graphics separation in maps. In: Proceedings of the 4th IAPR international workshop on graphics recognition, pp 167–177
3. Chen C-C, Knoblock CA, Shahabi C (2008) Automatically and accurately conflating raster maps with orthoimagery. GeoInformatica 12(3):377–410
4. Chen L-H, Wang J-Y (1997) A system for extracting and recognizing numeral strings on maps. In: Proceedings of the 4th international conference on document analysis and recognition, vol 1, pp 337–341
5. Chiang Y-Y, Knoblock CA, Shahabi C, Chen C-C (2009) Accurate and automatic extraction of road intersections from raster maps. GeoInformatica 13(2):121–157
6. Chiang Y-Y, Knoblock CA (2010) An approach for recognizing text labels in raster maps. In: Proceedings of the 20th international conference on pattern recognition, pp 3199–3202
7. Chiang Y-Y, Knoblock CA (2011) Recognition of multi-oriented, multi-sized, and curved text. In: Proceedings of the 11th international conference of document analysis and recognition, pp 1399–1403
8. Chiang Y-Y, Knoblock CA (2013) A general approach for extracting road vector data from raster maps. Int J Doc Anal Recog 16(1):55–81
9. Chiang Y-Y, Knoblock CA (2012) Generating named road vector data from raster maps. Geographic information science, lecture notes in computer science, vol 7478/2012, pp 57–71
10. Deseilligny MP, Mena HL, Stamonb G (1995) Character string recognition on maps, a rotation-invariant recognition method. Pattern Recog Lett 16(12):1297–1310
11. Edmondson S, Christensen J, Marks J, Shieber SM (1996) A general cartographic labelling algorithm. Cartographica Int J Geogr Inf Geovisualization 33(4):13–24
12. Fletcher LA, Kasturi R (1988) A robust algorithm for text string separation from mixed text/graphics images. IEEE Trans Pattern Anal Mach Intell 10(6):910–918
13. Gelbukh A, Levachkine S, Han S-Y (2004) Resolving ambiguities in toponym recognition in cartographic maps. In: Proceedings of the 5th IAPR international workshop on graphics recognition, pp 104–112
14. Goto H, Aso H (1998) Extracting curved text lines using local linearity of the text line. Int J Doc Anal Recognit 2(2–3):111–119
15. Kanai J, Rice SV, Nartker TA, Nagy G (1995) Automated evaluation of OCR zoning. IEEE Trans Pattern Anal Mach Intell 17(1):86–90
16. Leyk S, Boesch R (2010) Colors of the past: color image segmentation in historical topographic maps based on homogeneity. GeoInformatica 14(1):1–21
17. Li L, Nagy G, Samal A, Seth SC, Xu Y (2000) Integrated text and line-art extraction from a topographic map. Int J Doc Anal Recog 2(4):177–185
18. Li Y, Sun J, Tang C-K, Shum H-Y (2004) Lazy snapping. ACM Trans Graph 23(3):303–308
19. Mao S, Rosenfeld A, Kanungo T (2003) Document structure analysis algorithms: a literature survey. In: Proceedings of the SPIE conference on document recognition and retrieval X, vol 5010, pp 197–207

20. Mori S, Suen CY, Yamamoto K (1992) Historical review of OCR research and development. Proc IEEE 80(7):1029–1058. doi:10.1109/5.156468
21. Myers GK, Mulgaonkar PG, Chen C-H, DeCurtins JL, Chen E (1996) Verification-based approach for automated text and feature extraction from raster-scanned maps. In: Lecture notes in computer science, vol 1072. Springer, pp 190–203
22. Nagy G, Samal A, Seth S, Fisher T, Guthmann E, Kalafala K, Li L, Sivasubramaniam S, Xu Y (1997) Reading street names from maps - technical challenges. In: GIS/LIS conference, pp 89–97
23. Nagy GL, Nartker TA, Rice SV (2000) Optical character recognition: An illustrated guide to the frontier. In: Proceedings of the SPIE international symposium on electronic imaging science and technology, vol 3967, pp 58–69
24. Najman L (2004) Using mathematical morphology for document skew estimation. In: Proceedings of the SPIE conference on document recognition and retrieval IX, pp 182–191
25. Pal U, Sinha S, Chaudhuri BB (2003) Multi-oriented english text line identification. In: Proceedings of the 13th scandinavian conference on image analysis, pp 1146–1153
26. Pouderoux J, Gonzato JC, Pereira A, Guitton P (2007) Toponym recognition in scanned color topographic maps. In: Proceedings of the 9th international conference on document analysis and recognition, vol 1, pp 531–535
27. Rother C, Kolmogorov V, Blake A (2004) GrabCut: interactive foreground extraction using iterated graph cuts. ACM Trans Graph 23(3):309–314
28. Roy PP, Pal U, Lladós J, Kimura F (2008) Multi-oriented english text line extraction using background and foreground information. In: The eighth IAPR international workshop on document analysis systems, DAS '08, pp 315–322. doi:10.1109/DAS.2008.83
29. Roy PP, Pal U, Lladós J, Delalandre M (2009) Multi-oriented and multi-sized touching character segmentation using dynamic programming. In: Proceedings of the 10th international conference on document analysis and recognition, pp 11–15
30. Velázquez A, Levachkine S (2004) Text/graphics separation and recognition in raster-scanned color cartographic maps. In: Lladós J, Kwon Y-B (eds) Graphics recognition of lecture notes in computer science, vol 3088. Springer, pp 63–74
31. Wong KY, Wahl FM (1982) Document analysis system. IBM J Res Dev 26:647–656

Yao-Yi Chiang, Ph.D., is an Assistant Professor (Research) in Spatial Sciences in the Spatial Sciences Institute, University of Southern California (USC). He is also a Visiting Computer Scientist at the USC Information Sciences Institute. Dr. Chiang received his Master and Ph.D. degrees in Computer Science from University of Southern California in 2004 and 2010, respectively; his Bachelor degree in Information Management from the National Taiwan University in 2000. His general area of research is information integration, with a focus on discovery, extraction, and fusion of geospatial data from heterogeneous sources. Dr. Chiang is an expert in digital map processing, pattern recognition, and geospatial information system (GIS). He teaches spatial databases and GIS programing. He has published many articles on automatic techniques for geospatial data extraction and integration. Prior to USC, Dr. Chiang worked as a research scientist for Geosemble Technologies (now TerraGo Technologies), which was founded based on a patent on geospatial-data fusion techniques and he was a co-inventor.

Craig Knoblock is a Research Professor in Computer Science at the University of Southern California (USC) and the Director of Information Integration at the USC Information Sciences Institute. He received his Bachelor of Science degree from Syracuse University, and his Master's and Ph.D. from Carnegie Mellon University, all in computer science. Dr. Knoblock is also a founder of Fetch Technologies, a web extraction and integration provider, and of Geosemble Technologies, which develops geospatial data integration solutions. At USC, Dr. Knoblock leads a team of about 20 researchers, staff and students in developing techniques for rapid, efficient information integration. He focuses on constructing distributed, integrated applications from online sources through information extraction, source modeling, record linkage, constraint reasoning and other technologies for geospatial and bioinformatics data integration.

Dr. Knoblock is a Fellow of the Association for the Advancement of Artificial Intelligence (AAAI), a Distinguished Scientist of the Association of Computing Machinery (ACM), President and Trustee of the International Joint Conference on Artificial Intelligence (IJCAI), and past President of the International Conference on Automated Planning and Scheduling (ICAPS). He has served on the Senior Program Committee of the National Artificial Intelligence Conference, among others, and is conference chair for the 2011 International Joint Conference on AI (IJCAI). Dr. Knoblock has published Generating Abstraction Hierarchies (Kluwer Academic Publishers, 1993), along with more than 200 journal articles, book chapters and conference papers. He serves on the Editorial Boards of several journals, including Artificial Intelligence and the Journal of Web Semantics.