Geoinformatica
DOI 10.1007/s10707-008-0046-3

Automatic and Accurate Extraction of Road Intersections from Raster Maps

Yao-Yi Chiang · Craig A. Knoblock · Cyrus Shahabi · Ching-Chien Chen

Received: 23 June 2006 / Revised: 19 December 2007 /
Accepted: 4 February 2008

Abstract Since maps are widely available for many areas around the globe, they provide a valuable resource to help understand other geospatial sources such as to identify roads or to annotate buildings in imagery. To utilize the maps for understanding other geospatial sources, one of the most valuable types of information we need from the map is the road network, because the roads are common features used across different geospatial data sets. Specifically, the set of road intersections of the map provides key information about the road network, which includes the location of the road junctions, the number of roads that meet at the intersections (i.e., connectivity), and the orientations of these roads. The set of road intersections helps to identify roads on imagery by serving as initial seed templates to locate road pixels. Moreover, a conflation system can use the road intersections as reference features (i.e., control point set) to align the map with other geospatial sources, such as

This work is based on an earlier paper: Automatic Extraction of Road Intersections from Raster Maps, in Proceedings of the 13th annual ACM International Symposium on Geographic Information Systems, Pages: 267–276, ©ACM, 2005. http://doi.acm.org/10.1145/1097064.1097102.

Y.-Y. Chiang (✉) · C. A. Knoblock
Department of Computer Science and Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292 USA
e-mail: yaoyichi@isi.edu

C. A. Knoblock
e-mail: knoblock@isi.edu

C. Shahabi
Computer Science Department, University of Southern California, SAL 300, Los Angeles, CA 90089-0781, USA
e-mail: shahabi@usc.edu

C.-C. Chen
Geosemble Technologies, 2041 Rosecrans Ave. Suite 245 El Segundo, CA 90245 USA
e-mail: jchen@geosemble.com

aerial imagery or vector data. In this paper, we present a framework for automatically and accurately extracting road intersections from raster maps. Identifying the road intersections is difficult because raster maps typically contain much information such as roads, symbols, characters, or even contour lines. We combine a variety of image processing and graphics recognition methods to automatically separate roads from the raster map and then extract the road intersections. The extracted information includes a set of road intersection positions, the road connectivity, and road orientations. For the problem of road intersection extraction, our approach achieves over 95% precision (correctness) with over 75% recall (completeness) on average on a set of 70 raster maps from a variety of sources.

Keywords Raster map · Road layer · Road intersection · Imagery · Conflation · Fusion · Vector data · Geospatial data integration

1 Introduction

Humans have a very long history of using maps. Fortunately, due to the widespread use of Geographic Information System and the availability of high quality scanners, we can now obtain many maps in raster format from various sources, such as digitally scanned maps from USGS Topographic Maps on Microsoft Terraserver[1] or computer generated maps from U.S Census Bureau[2] (TIGER/Line Maps), Google Maps,[3] Yahoo Maps,[4] MapQuest Maps,[5] etc. Not only are the raster maps readily accessible compared to other geospatial data (e.g., vector data), but they provide very rich information, such as road names, landmarks, or even contour lines. Consider the fact that road maps are generally the most frequently used map type and the road layers commonly exist across many different geospatial data sources, the road network can be used to fuse a map with other geospatial data. In particular, instead of extracting the whole road network, road intersection templates (i.e., the positions of the road intersections, the road connectivity and orientations) provide the basic elements of the road layout and are comparatively easy to extract. For example, the road intersection templates can be used to extract roads from other sources, such as aerial images [13]. As shown in Fig. 1, without using the whole road network, the extracted road intersection templates can be used as seed templates to extract the roads from the imagery of Tehran, Iran where we have limited access to the vector data. Moreover, since the layout of a set of road intersections within a certain area is usually unique, we can utilize the road intersections as the "fingerprint" of the raster map to determine the extent of a map that is not georeferenced [9]; and further, the road intersections can be used as feature points to align the raster map with other geospatial sources [5]. For example, Fig. 2a shows a hybrid view of Tehran, Iran from Google Map where only major roads are shown with labels. By matching the layout

[1] http://terraserver-usa.com/

[2] http://tiger.census.gov/cgi-bin/mapsurfer

[3] http://map.google.com

[4] http://map.yahoo.com

[5] http://www.mapquest.com

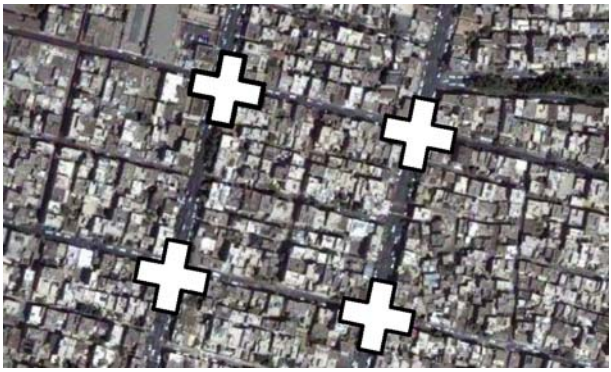

Fig. 1 Use the extracted road intersections as seed templates to extract roads from imagery

of road intersections from a tourist map found on the Internet shown in Fig. 2b to the corresponding layout on the satellite imagery, we can create a enhanced integrated representation of the maps and imagery as shown in Fig. 3.

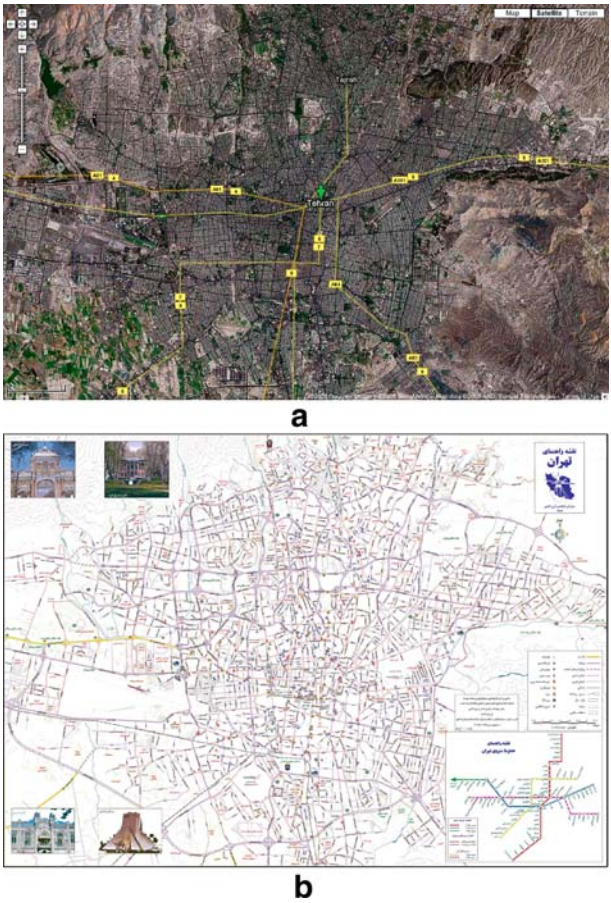

Fig. 2 The integration of imagery from Google Map and a tourist map (Tehran, Iran). **a** The Google Map hybrid view. **b** The tourist map in Farsi

Fig. 3 The integration of imagery from Google Map and a tourist map (Tehran, Iran)

Automatic extraction of road intersections is difficult due(e.g., map geocoordinates, scales, legend, original vector data, etc) and also the great complexity of maps. For example, for a Google Map snapshot in an article or an evacuation map posted on a wildfire news website, there is no uniform metadata or auxiliary information that can be automatically found and parsed to help utilize the map without human intervention. In addition, maps typically contain multiple layers of roads, buildings, symbols, characters, etc. People interpret the maps by examining the map context such as road labels or looking for the map legends, which is a difficult task for machines. To overcome these problems, we present a framework, which works on the pixel level and utilizes the distinctive geometric properties of road lines and intersections to automatically extract road intersections from raster maps. The inputs of our approach are raster maps from a variety of sources without any auxiliary information such as the color of the roads, vector data, or legend information, etc.

Fig. 4 Example input raster maps. **a** USGS Topographic Map. **b** TIGER/Line Map

Two examples of the input raster maps are shown in Fig. 4. The outputs are the positions of the road intersection points, the number of roads that meet at each intersection as well as the orientation of each intersected road.

For the problem of automatic extraction of road intersections from raster maps, much of the previous work provides partial solutions [2], [6], [10], [11], [15]–[18], [24]–[26]. A majority of the work [2], [6], [15], [26] focuses on extracting text from maps and the road lines are typically ignored in the process. Others such as Salvatore and Guitton [24] work on extracting and rebuilding the lines; however they rely heavily on prior learned colors to separate the lines for specific data sets, which is not generally useful in our automatic process. Habib et al. [11] assume the maps contain only roads, and work on road lines directly to extract intersections. These previous approaches require manual interactions as they provide only partial solutions; and we build on some of the previous work in our framework to solve specific sub-problems. For example, to extract road lines from the foreground pixels, we incorporate Cao et al.'s algorithm [6] as the first step to remove small connected objects such as characters from the map. Our approach is a complete solution that does not rely on prior knowledge of the input raster maps.

This paper is based on our earlier paper [7] of this work, and the new contributions of this paper are as follows:

1. We describe our approach in more detail (Section 2).
2. We describe how we utilize the Localized Template Matching (LTM) [4] to improve our algorithm (Section 2.6.3). We also report the similarity value generated by LTM to evaluate the quality of the extracted road intersection templates (i.e., the road intersection, connectivity and orientations) (Section 3).
3. We present a more extensive set of experiments testing our approach on more map sources, and we also report the computation time with respect to the number of foreground pixels on a map (Section 3).

4. We report our precision, recall, and F-measure on different positional accuracy levels (Section 3).
5. We report the results of a comparison on the same set of map sources using the approach in this paper against the approaches in our previous work [3] and an earlier paper of this work [7] (Section 3).

The remainder of this paper is organized as follows. Section 2 describes our approach to automatically extract road intersections. Section 3 reports on our experimental results. Section 4 discusses the related work and Section 5 presents the conclusion and future work.

2 Automatic road intersection extraction

The overall approach described in this paper is shown in Fig. 5. There are three major steps in our approach:

1. **Automatic Segmentation** We first extract the foreground pixels from the raster map as shown in Fig. 6a and Fig. 6b.
2. **Pre-Processing – Extract and Rebuild Road Layer** After we obtain the foreground pixels, we further extract the road pixels by removing pixels which do not hold the properties to constitute roads (e.g., pixels for labels, contour lines, etc) as shown in Fig. 6c. Then we utilize a number of image processing operators to rebuild the extracted roads as shown in Fig. 6d.
3. **Determine Road Intersections and Extract Connectivity with Road Orientation** With the extracted roads, we detect possible road intersection candidates and utilize the number of roads that meet at an intersection candidate (i.e., connectivity) and the road orientations to determine an actual road intersection. The extracted road templates are shown as black thick lines in Fig. 6e.

The following sub-sections describe each component in detail.

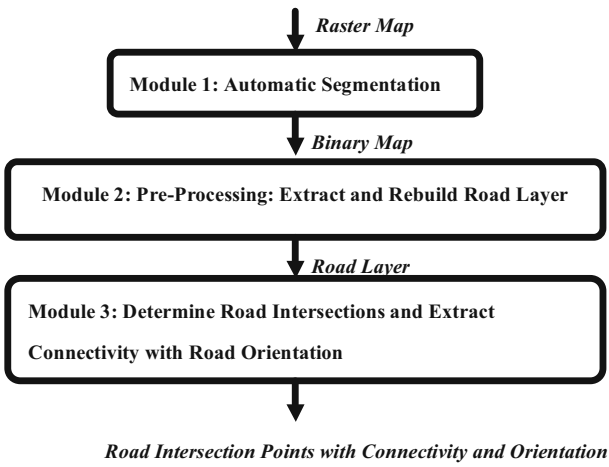

Fig. 5 Approach to extract road intersections

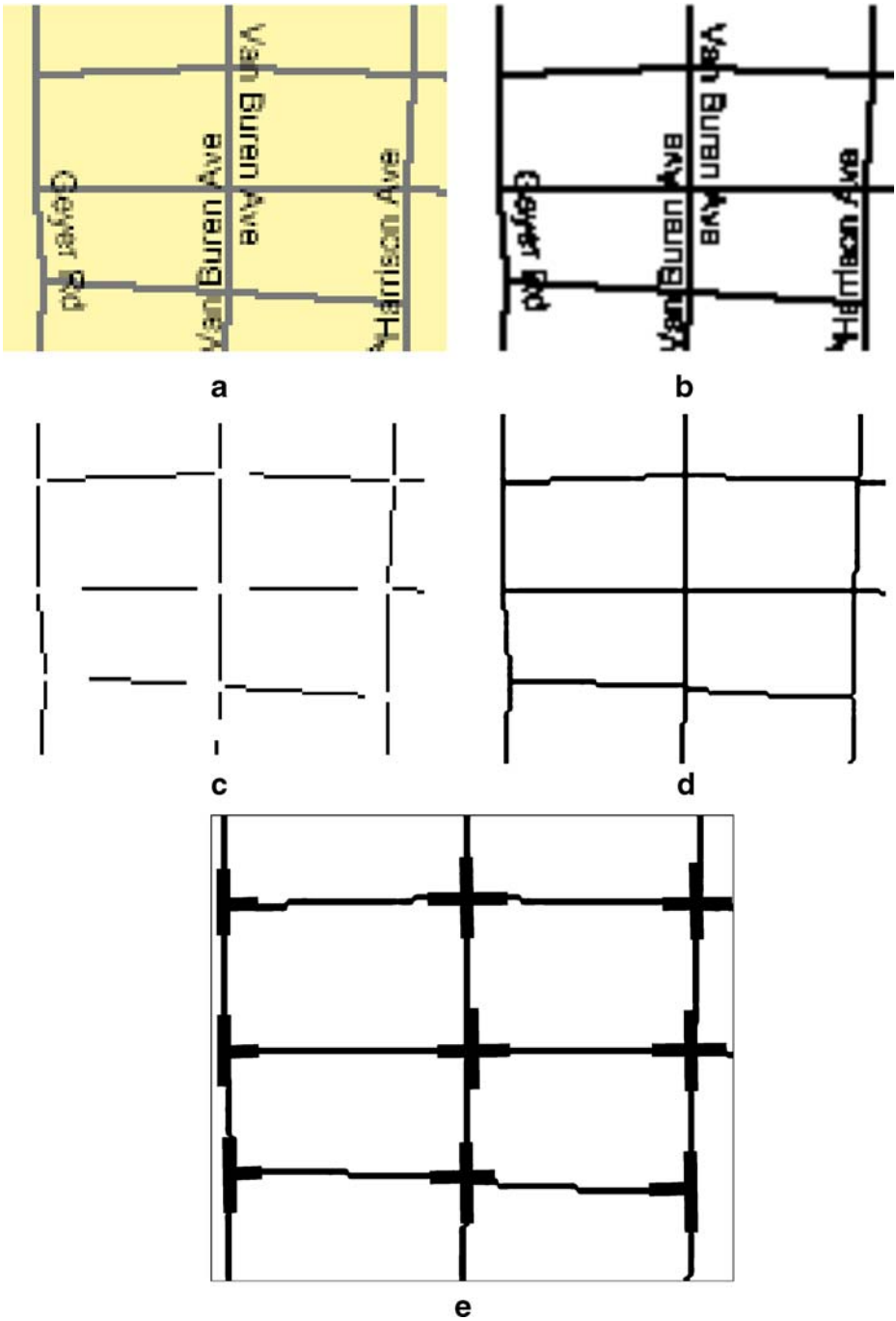

Fig. 6 Remove the background then extract and rebuild the road layer from the raster map. **a** Original map. **b** Foreground objects. **c** Broken lines. **d** Restored lines. **e** Extract Intersections

2.1 Automatic segmentation

In this step, the input is a raster map, and our goal is to extract the foreground pixels from the map. We utilize a technique called segmentation with automatically generated thresholds to separate the foreground from the background pixels. Many segmentation methods are discussed and proposed for various applications [23], and we implement a method that analyze the shape of the grayscale histogram and classify the histogram clusters based on their sizes. Our implementation is based on our observations of the color usage on raster maps, which are:

1. The background colors of raster maps have a dominant number of pixels; and the number of foreground pixels are significantly smaller than the number of foreground background pixels.
2. The foreground colors have high contrast against the background colors.

The first step is to convert the original input raster map to an 8-bit grayscale (i.e., 256 luminosity levels) image as shown in Fig. 7a. The conversion is done by computing the average value of the red, green and blue strength of each pixel for the grayscale level. The grayscale histogram of Fig. 7a is shown in Fig. 8. In the histogram, the X-axis represents the luminosity level; from 0 (black) to 255 (white), and the Y-axis has the histogram values that represent the number of pixels for each luminosity level. We then partition the histogram into luminosity clusters and label the clusters as either background or foreground clusters.

Since the background colors have a dominant number of pixels, we start from searching for the the global maximum on the histogram value to identify the first background cluster. As shown in Fig. 8, we first find *PEAK 1*, then we check whether

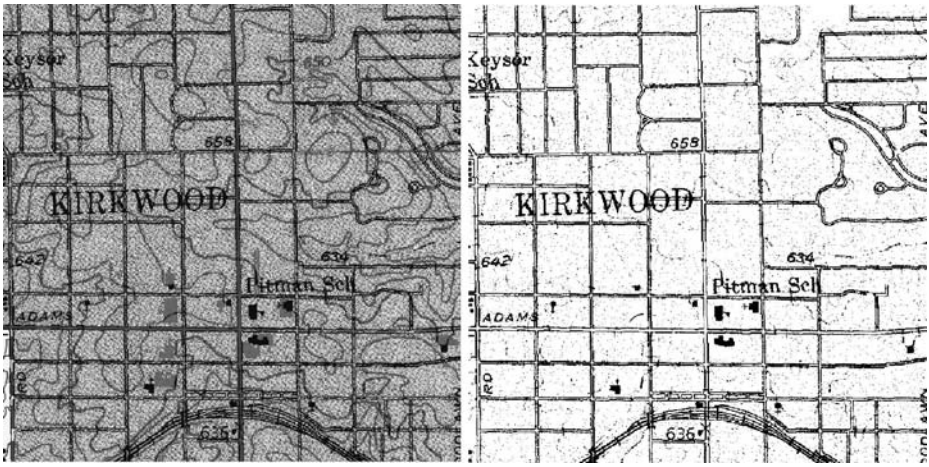

Fig. 7 Before and after automatic segmentation (USGS Topographic Map). **a** Before (grayscale map). **b** After (binary map)

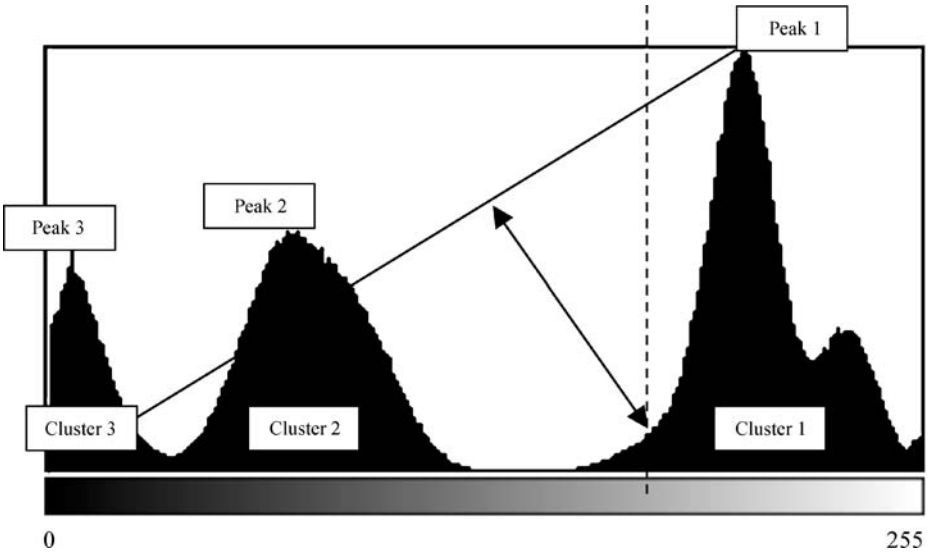

Fig. 8 Find *Cluster 1*'s left bound threshold using the triangle method

PEAK 1 is closer to 255 (white) or 0 (black) in the histogram to determine which part in the histogram (i.e., the portion of histogram to the left or right of *PEAK 1*) contains the foreground colors. If the peak is closer to 255, such as *PEAK 1*, 255 (white) is used as the right boundary of the first background cluster. This is because the foreground colors generally have high contrast against the background colors. For example, if a light gray color of luminosity 200 is used as the background, it is common to find the foreground colors spread in the histogram between 0 to 200 instead of between 200 to 255. As a result, for *Cluster 1* in Fig. 8, we identify 255 as its right boundary, and its left boundary is located using the triangle method proposed by Zack [27], which will be discussed latter. On the other hand, if *PEAK 3* is the global maximum, we will use 0 as its left boundary, and its right boundary is then located using the triangle method.

After we find the first cluster, we search for next peak and use the triangle method to locate the cluster's left and right boundaries until every luminosity level in the histogram belongs to some cluster. Any clustering algorithm could be used to find the cluster boundary, and we use the triangle method proposed by Zack [27] for its simplicity. As shown in Fig. 8, to find the left or right boundary, we construct a line called triangle line between the peak and the 0 or 255 end point and compute the distance between each Y-value and the triangle line. The luminosity level that has its Y-value under the triangle line and has the maximum distance is the cluster's left/right boundary as the dotted line indicates in Fig. 8. If every luminosity level has its Y-value above the triangle line, then the 0 or 255 end point is used as the left or right boundary of the cluster. For example, if we try to find the left boundary for *Cluster 3* using the triangle method, we will find that every Y-value on the left of *PEAK 3* is above the triangle line, so 0 is used for the left boundary of *Cluster 3*.

Table 1 Number of pixels in each cluster

	Cluster 1	Cluster 2	Cluster 3
Cluster pixels	316846	237876	85278
Cluster pixels / total pixels	50%	37%	13%

In our example, *Cluster 1* is first classified as the first background cluster. The foreground colors usually have high contrast against the background colors, so the cluster which is the farthest from the first background cluster in the histogram is the first foreground cluster (i.e., *Cluster 3* in our example). For the remaining clusters, we compare the number of pixels in each cluster to the background cluster and the foreground cluster. If the number of pixels of a cluster is closer to the number of pixels of the background clusters than foreground clusters, it is classified as a background cluster; otherwise it is a foreground cluster. For example, as shown in Table 1, the number of pixels in *Cluster 2* is closer to *Cluster 1*(background) than *Cluster 3* (foreground), so *Cluster 2* is classified as a background cluster. The idea is, if a cluster uses as many pixels as any of the background cluster, we do not want to include it in our result for two reasons. First, the cluster could be another color used in the background. Second, the cluster could be the color used to fill up large objects on the map such as parks, lakes, etc. Since our goal is to look for road lines in the foreground pixels, the cluster should be discarded in either case. After each cluster is classified as background or foreground, we remove the background clusters (i.e., convert the color of pixels in background clusters to white). An example of the resulting binary image is shown in Fig. 7b.

There is no universal solution for the automatic segmentation problem. An edge detector is sometimes used instead of thresholding [11]; however, the edge detector is sensitive to noise, which makes it difficult to use for our approach. Moreover, with the presence of characters, the edge detector makes the resulting characters bigger than the original ones since the edge detector produces two edges for a character line segment. Big characters have more overlap with road lines and hence severely sabotage the results of the next step that extract and rebuild the road layer. On the other hand, a simple thresholding algorithm can be applied to a wide-range of map sources.

2.2 Pre-processing—extracting and rebuilding road layers

After we separate the foreground from the background pixels, we have a binary raster map that contains multiple layers such as roads, labels, etc. In this module, we extract and rebuild the road layer from the binary map. In order to extract the road layer, we need to remove the objects that do not possess the geometric properties of roads. Without losing generality, our assumptions about roads on raster maps are:

1. Road lines are straight within a small distance (i.e., several meters).
2. The linear structures (i.e., lines) in a map are mainly roads. The majority of the roads share the same road format—single-line roads or double-line roads, and the double-line roads have the same road width within a map. Examples of single-line and double-line maps are shown in Fig. 9

Fig. 9 Single-line and double-line maps. **a** Single-line map from TIGER/Line Map. **b** Double-line map from Google Map

3. Unlike label layers or building layers, which could have many small connected objects, road lines are connected to each other as road networks and road layers usually have a small number of connected objects or even only one large connected object – the whole road layer is connected.

Based on these assumptions, we classify the map into two format, the single-line map and double-line map, automatically by detecting the format of the majority of the roads on the map. If a map is classified as double-line road format (e.g., Google Map in Fig. 9b), we trace the double-line roads and remove single-line objects (e.g., contour lines, rivers, railroads, etc) from the map. For other maps that have the majority roads in single-line format (e.g., TIGER/Line map in Fig. 9a), we classify them as single-line maps and all linear structures on the maps are preserved for further processing. Since we classify the maps into either double-line map or single-line map, we might lose some roads in maps with mainly double-line roads and some single-line roads. However, by identifying the major road format we dramatically removed many unwanted lines such as the contour lines. After we detect the road format and process the map, we remove text labels from the remaining foreground pixels and reconnect the broken lines. The following subsections describe our algorithm to automatically check the road layer format, to trace the road lines, and to rebuild the road layer.

2.2.1 Double-line format detection and parallel-pattern tracing

To detect the road format, we need to differentiate the geometric properties of single-line and double-line roads. We first define one-pixel width straight lines in the grid domain (i.e., the raster format). As shown in Fig. 10, there are four possible one-pixel width straight lines constituted by pixel C's eight adjacent pixels, which represent the lines of 0°, 45°, 90°, and 135° respectively. These four lines are the basic

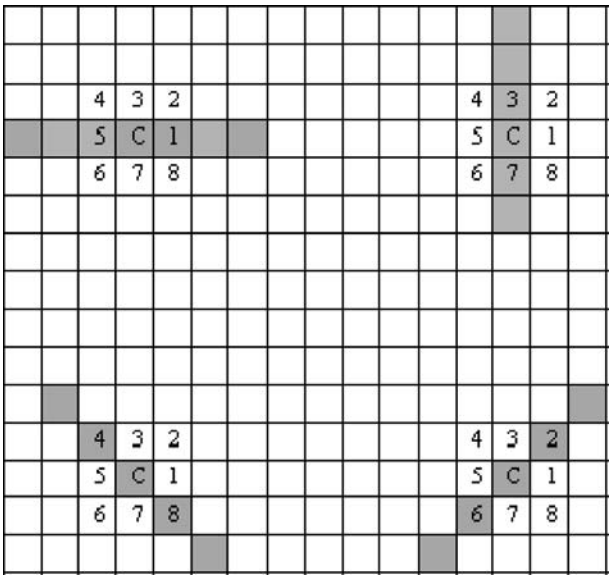

Fig. 10 Basic single-line road elements

elements to constitute other straight lines with various slopes starting from pixel *C*. For example, in Fig. 11, to represent a 30° line, one of the basic elements, a 45° line segment, is drawn in black, then additional gray pixels is added to tilt the line. Hence, by identifying these basic elements, we can identify straight lines in single-line format.Based on the four basic elements, if the pixel *C* is on a double-line roads shown in Fig. 12, there are eight different parallel-patterns of double-line roads (i.e., the dashed cells are possible parallel lines). If we can detect any of the eight patterns for a given foreground pixel, we classify the pixel as a double-line road pixel. As shown in Fig. 12, we draw a cross at the pixel *C* using the size of the road width; and the cross locates at least two road line pixels on each pattern, one at the horizontal direction and the other one at the vertical direction. Therefore, if we know the road

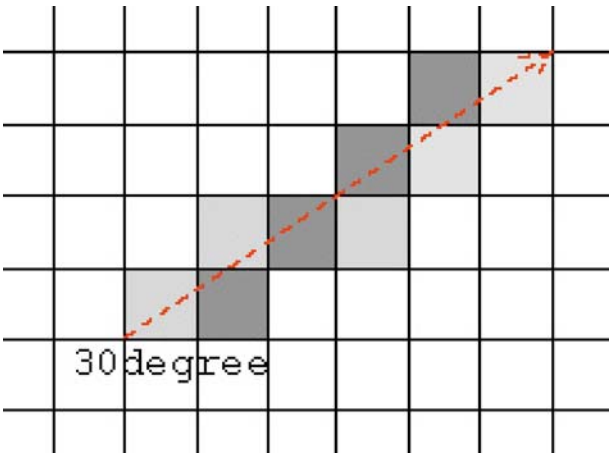

Fig. 11 A 45° basic element (*black*) in the 30° line segment (both *black* and *gray*)

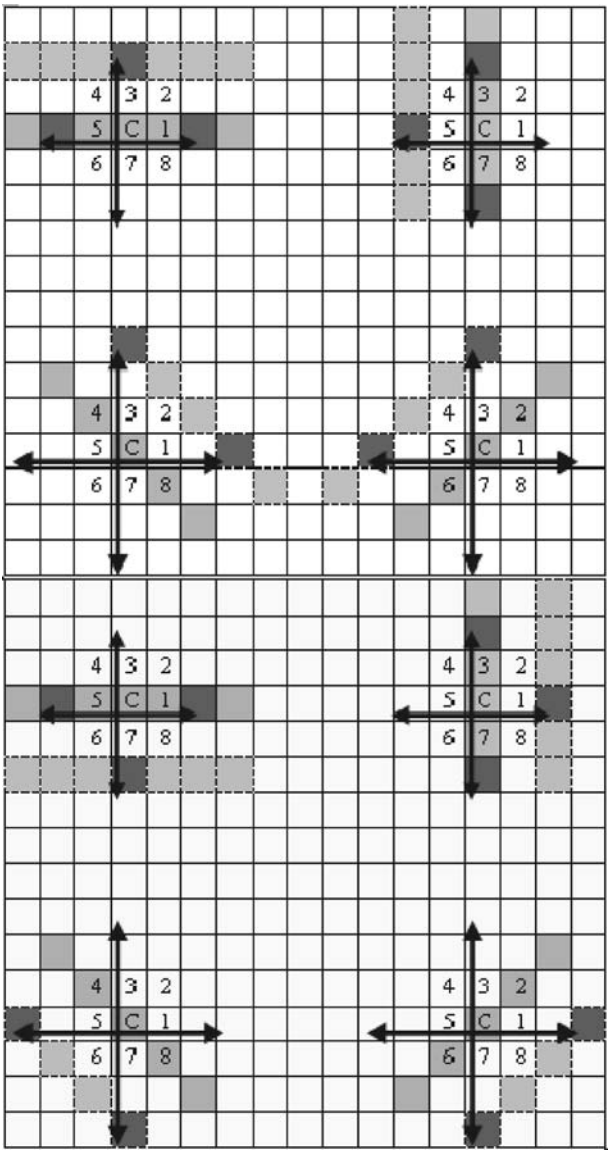

Fig. 12 Basic double-line road elements

width, we can determine whether a foreground pixel is on a double-line road layer by searching for the corresponding foreground pixels within a distance of road width (RW)[6] in vertical and horizontal directions to find the parallel pattern. Two examples are shown in Fig. 13. If a foreground pixel is on a horizontal or vertical road line as shown in Fig. 13a, we can find two foreground pixels along the orientation of the road line within a distance of RW and at least another foreground pixel on the corresponding parallel road line in a distance of RW. If the orientation of the road

[6] RW is used in this paper as a variable representing the road width in pixels.

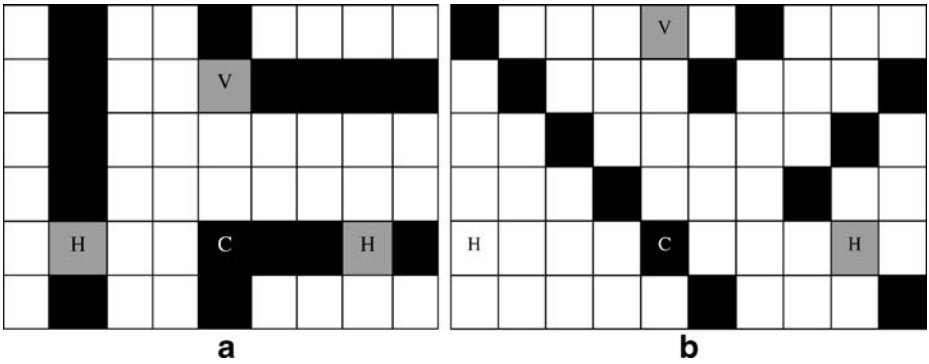

Fig. 13 Double-line format checking and parallel-pattern tracing. **a** RW = 3. **b** RW = 4

line is neither horizontal nor vertical as shown in Fig. 13b, we can find one foreground pixel on both horizontal and vertical directions on the corresponding parallel road line in a distance of *RW* as shown in Fig. 13b. By tracing the parallel pattern, we are able to detect double-line road pixels; however, we still need to know:

1. The format (i.e., single-line or double-line format) of the input road layer. If it is a single-line road layer, we skip this procedure.
2. The road width (*RW*).

To detect the road layer format, we apply the parallel-pattern tracing algorithm on the binary map varying the road width from one to *M* pixels and remove foreground pixels which are not classified as road line pixels for a given road width. *M* is the maximum width in pixel of a double-line road we expect from the map. For example, on a 2 m/pixel map given *M* as 10 pixels, 20 m wide double-line roads are the widest double-line road we want to detect from the map. It is an input parameter for the user to fine-tune the algorithm. A bigger *M* ensures that the roads on high resolution maps can be found, but it requires more processing time. Since we do not know the map resolution, we use *M* equal to 10 in the following examples in this section.

After we remove foreground pixels that are not detected as a road line pixel for the road width 1 to *M*, we then compute the ratio of the remaining foreground pixels divided by the original foreground pixels for each road width. For a given *RW*, if the majority of foreground pixels possess the double-line property, we classify the raster map as a double-line map and the road width is *RW*. Figure 14 shows the results of this process of various double-line and single-line maps. At the beginning of this process when the road width is one pixel, the remaining foreground pixel ratios are close to one (100%). After *RW* increases, the ratio starts to decline. This is because foreground pixels tend to be near each other, and it is easier to find corresponding pixels even if the road width is incorrect or the map is not a double-line map. The peaks in the chart shown in Fig. 14a imply that the input raster maps have double-line road layer and the road width is correct since most of the road pixels are not removed at the correct road width. For example, the high resolution ESRI Map has a peak at two pixels, the high resolution MapQuest Map, high resolution Yahoo Map, and USGS Topographic Map have a peak at four pixels in the chart as shown in Fig. 14a. ESRI Maps, MapQuest Maps which are not high resolution, and the TIGER/Line Maps are all single line maps, which do not have any peak as shown in Fig. 14b.

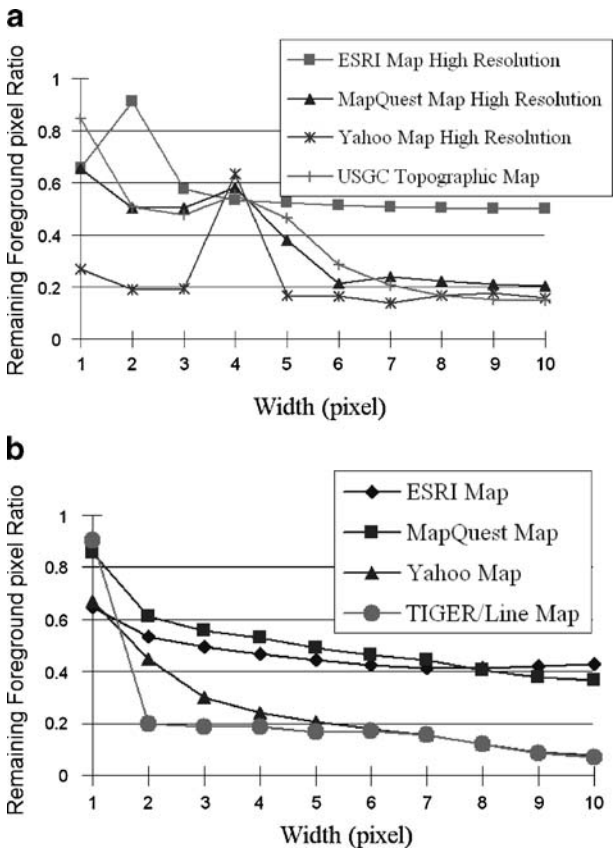

Fig. 14 Double-line format checking. **a** Double-line format road layers. **b** Single-line format road layers

Using this method, we detect road layers in double-line format automatically and also obtain the road width by searching for the peak. For example, from Fig. 14a, we know the USGS Topographic map is a double-line map with road width equal to 4 pixels. Hence, we apply the parallel-pattern tracing algorithm setting *RW* to 4 pixels. The resulting image is shown in Fig. 15 with the contour lines removed.

There are some exceptions to use parallel-pattern tracing to trace double-line pixels. As shown in Fig. 16, foreground pixels 1 to 8 are the example pixels which will be classified as a road line pixel using our parallel-pattern tracing algorithm, while pixels *A* to *D* are the example pixels which belong to the double-line road layer but will not be classified as a road line pixel. In Fig. 16, gray pixels are the corresponding pixels of pixels 1 to 8 in horizontal/vertical direction or on the corresponding parallel lines. Although after the parallel-pattern tracing, pixels *A* to *D* will be removed resulting in gaps between line segments, the majority of road pixels are detected and the gaps will be fixed later when we rebuild the whole road layer.

2.2.2 Text/graphics separation

After we detect the road format and process the map according to its format, the remaining noise comes from the small connected objects, e.g. buildings, symbols,

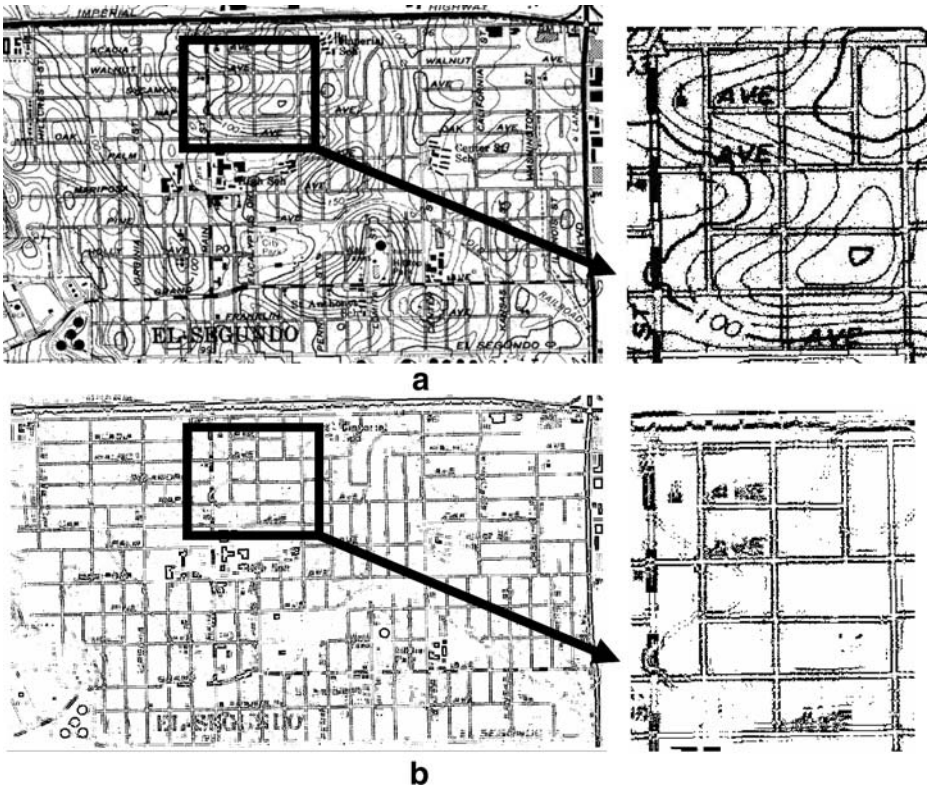

Fig. 15 USGS Topographic Map before and after parallel-pattern tracing. **a** Before. **b** After

characters, etc. The small connected objects tend to be near each other on the raster maps, such as characters that are close to each other to form strings, and buildings that are close to each other on a street block. The text/graphics separation algorithms [2], [6], [10], [15]–[18], [25], [26] are very suitable for grouping and removing these types of objects. The text/graphics separation algorithms start by identifying small connected objects and then use various algorithms to search neighborhood objects in order to build object groups [25].

We apply Cao et al.'s algorithm described in [6] for text/graphics separation. Their algorithm first removes small connected objects that do not overlap with other objects in the raster map as shown in Fig. 17b and then checks the length of each remaining line segment to determine if the line segment belongs to a graphic object. If a line segment is longer than a preset threshold, it is considered a graphic object (i.e., a line); otherwise, it is a text object (i.e., not a line). The identified text objects are shown in Fig. 17c, and the final result after text/graphics separation is show in Fig. 17d. The broken road lines are inevitable after the removal of those objects touching lines, and we can reconnect them when rebuilding the road layer.

With Cao et al.'s algorithm, we need to specify several parameters for the geometric properties of the characters, such as the size of one character and the

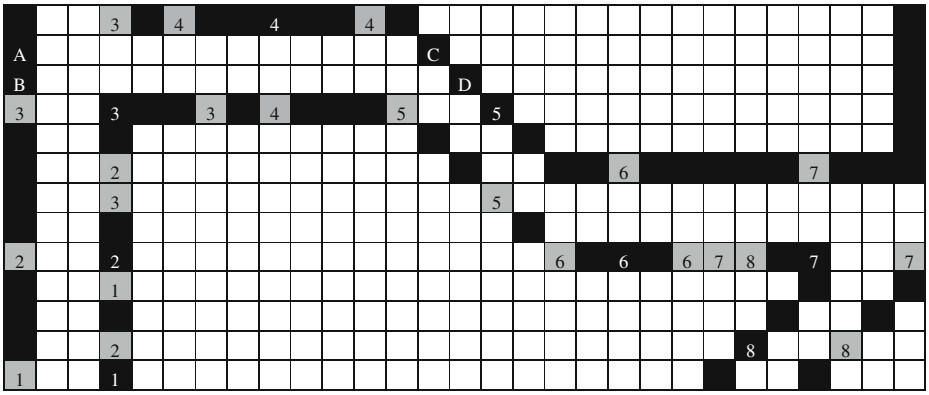

Fig. 16 The exceptions in double-line format checking and parallel-pattern tracing (*white cells* are background pixels)

maximum length of a word. Since we do not have the information to setup the algorithm for different maps, we first conduct several initial tests on a disjoint set of maps and select a set of parameters to be used in our experiments. Our experiment shows that Cao et al.'s algorithm is very robust since the input parameters do not have to be exact. Also, the characters sizes of many computer generated raster maps vary in a small range for users to read the map comfortably. For example, the maps from Google Map at different zooming level and maps from Yahoo Map have similar sizes of street labels. For the scanned maps, the characters could be enormously enlarged/shrunk depending on the scan resolution; and the text/graphics separation algorithm will fail given that no additional geometric properties of the characters are provided. However, scanned maps usually are scaled down for user-friendly viewing or to be displayed on the Internet, so it is not common that the input map is a scanned maps with a extremely high/low scan resolution and enormously large/small characters.

2.2.3 Rebuilding road layers: binary morphological operators

In the previous steps, we extracted the road layer and created broken lines during the extraction. In this step, we utilize the binary morphological operators to reconnect the lines and fix the gaps. Binary morphological operators are easily implemented using hit-or-miss transformations with various size masks [19], and are often used in various document analysis algorithms as fundamental operations [1]. The hit-or-miss transformation is performed in our approach as follows. We use 3-by-3 binary masks to scan over the input binary images. If the masks match the underlying pixels, it is a "hit"; otherwise, it is a "miss". Each operator uses different masks to perform hit-or-miss transformations and performs different actions as a result of a "hit" or "miss". We describe each operator in turn.

2.3 Binary dilation

The basic effect of a binary dilation operator is to expand the region of foreground pixels [19] and we use it to thicken the lines and reconnect adjacent pixels. As

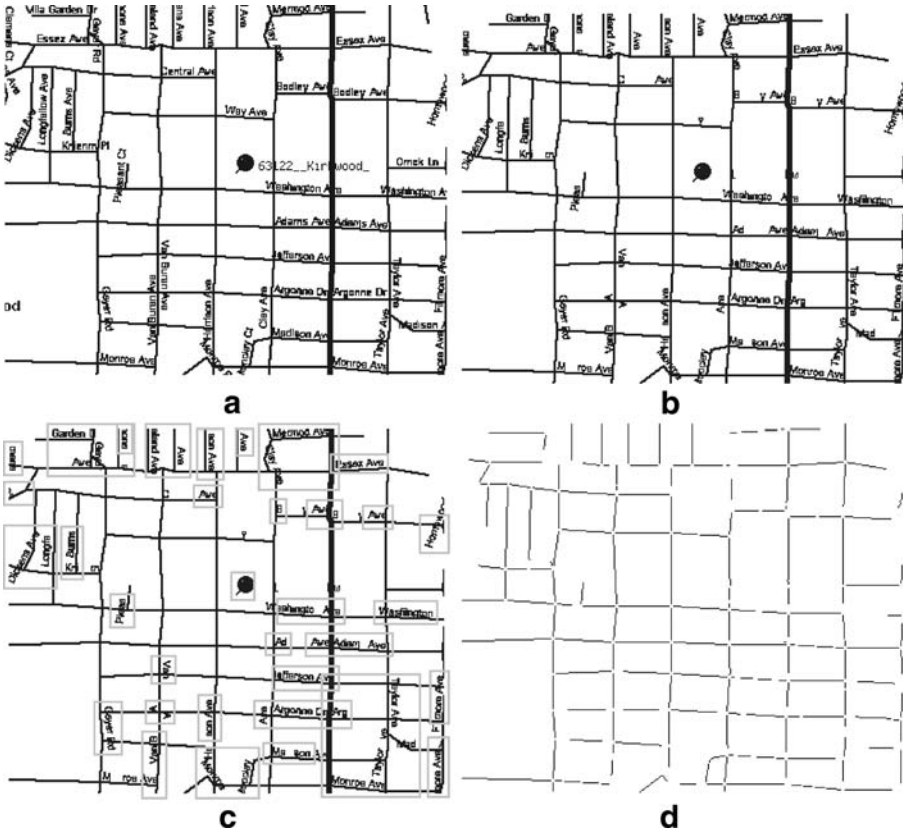

Fig. 17 TIGER/Line map before and after text/graphics separation. **a** Binary TIGER/Line map. **b** Remove small connected components. **c** Identify non-line objects. **d** After text/graphics separation

shown in Fig. 18, if a background pixel has any foreground pixel in its eight adjacent pixels (i.e., a "hit"), it is filled up as a foreground pixel (i.e., the action resulting from the "hit"). For example, Fig. 19 shows that after two iterations, the general dilation operator fixes the gap between two broken lines. Moreover, if the roads are in double-line format, the two parallel lines are combined to a single line after the

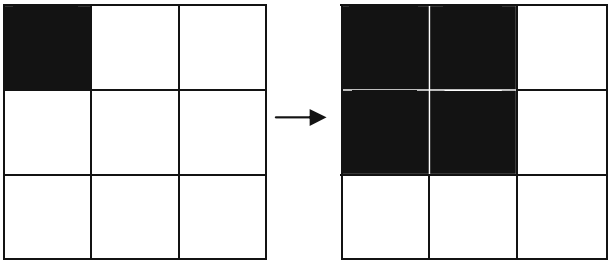

Fig. 18 Dilation (*black cells* are foreground pixels)

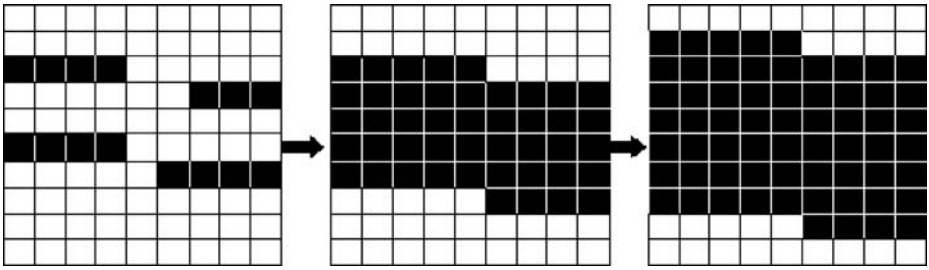

Fig. 19 The effect of the binary dilation operators (*black cells* are foreground pixels)

general dilation operator as shown in Fig. 19. The resulting image after performing three iterations of the binary dilation operator on Fig. 20a is shown in Fig. 20b. The number of iterations determines the maximum gap size that we can fix; the gaps smaller than six pixels are reconnected after performing three iterations of binary dilation operator and all road lines have become thicker. For double-line raster maps, the number of iterations is selected based on the width of the roads in order to merge the two parallel lines. For single-line maps, we utilize three iterations to fix gaps smaller than six pixels, which is chosen based on our initial experiments. Smaller number of iterations prevent two different roads from merging together but result in more broken lines; while larger number of iterations fix wide gaps but have the risk of merging two different roads.

2.4 Binary erosion

The idea of a binary erosion operator is to reduce the region of foreground pixels [19]. We use it to reduce the thick lines and maintain an orientation similar to the original orientation prior to applying the binary morphological operators. If a foreground pixel has any background pixel in its eight adjacent pixels (i.e., a "hit"),

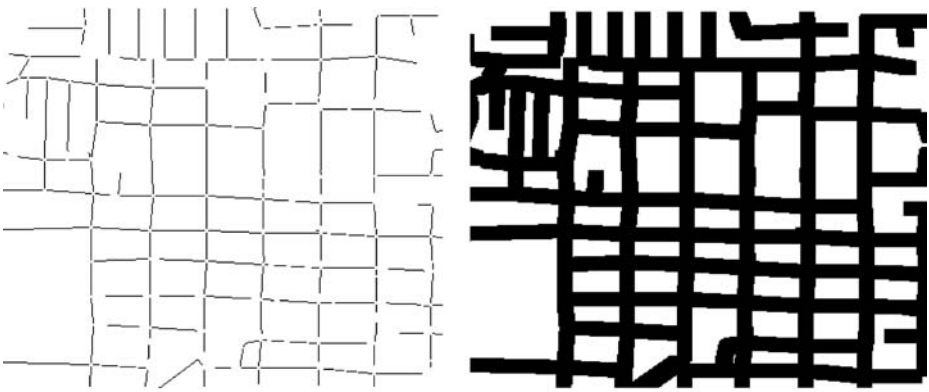

Fig. 20 TIGER/Line map before and after the binary dilation operator. **a** After text/graphics separation. **b** After the binary dilation operator

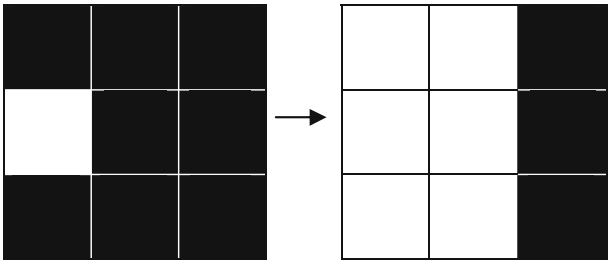

Fig. 21 Erosion operator (*black cells* are foreground pixels)

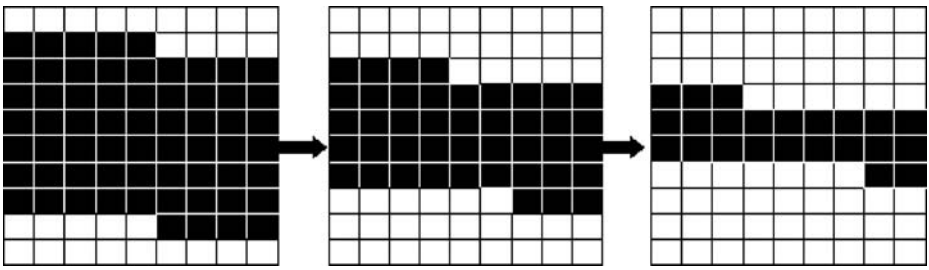

Fig. 22 The effect of the binary erosion operator (*black cells* are foreground pixels)

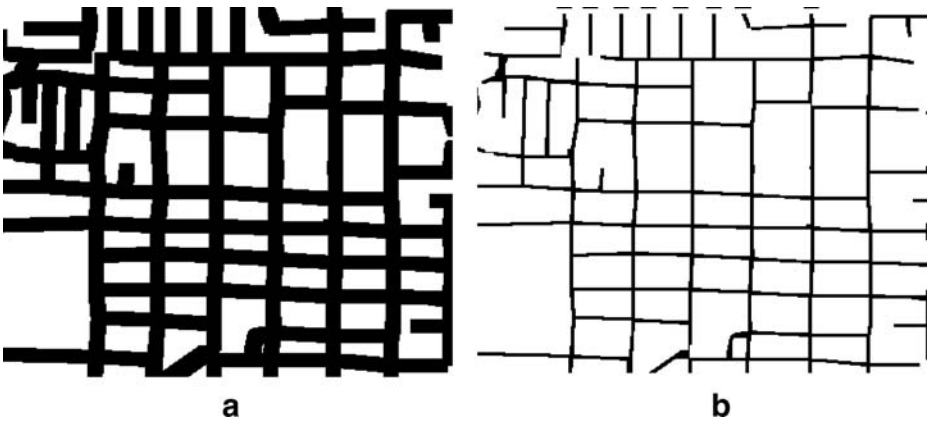

Fig. 23 TIGER/Line map before and after erosion. **a** After dilation. **b** After erosion

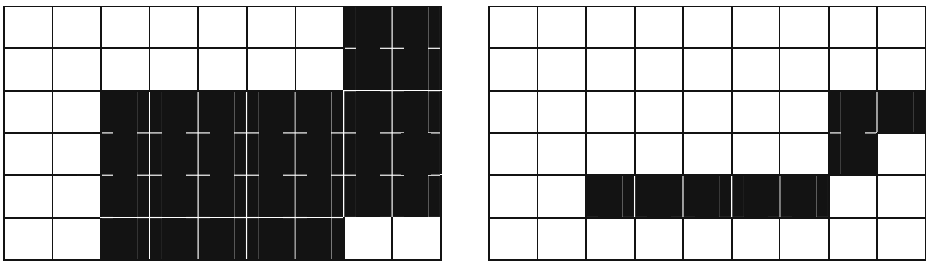

Fig. 24 The thinning operator (*black cells* are foreground pixels)

it is converted to a background pixel (i.e., the action resulting from the "hit") as shown in Fig. 21. For example, Fig. 22 shows that after two iterations, the general erosion operator reduces the width of the thick lines. The resulting image after performing two iterations of the binary erosion operator on Fig. 23a is shown in Fig. 23b.

2.5 Thinning

After applying the binary dilation and erosion operators, we have road layers composed from road lines with various widths. But we need the road lines to have exactly one pixel width to detect *interest points* and the connectivity in the next step, and the thinning operator can produce the one pixel width results. The effect of the thinning operator is shown in Fig. 24. We do not use the thinning operator right after the binary dilation operator because the binary erosion operator has the opposite affect to the binary dilation operator, which prevent the orientation of road lines near the intersections from being distorted, as shown in Fig. 25. We utilize a generic thinning operator that is a conditional erosion operator with an extra confirmation step [19]. The first step of the thinning operator is to mark every foreground pixel that connects to one or more background pixels (i.e., the same idea as the binary erosion operator) as candidate to be converted to the background. Then the confirmation step checks if the conversion of the candidate will cause any disappearance of original line branches to ensure the basic structure of the original objects will not be compromised. The resulting image after performing the thinning operator on Fig. 26a is shown in Fig. 26b.

2.6 Determine road intersections, connectivity, and orientation

In this step, we automatically extract road intersections, connectivity (i.e., the number of roads that meet at an intersection), and the orientation of the roads intersecting at each intersection from the road layer. In addition, we utilize the extracted information (i.e., the position of the extracted intersection point, connectivity, and orientation) and the original map to verify and improve the results in the last step of this module.

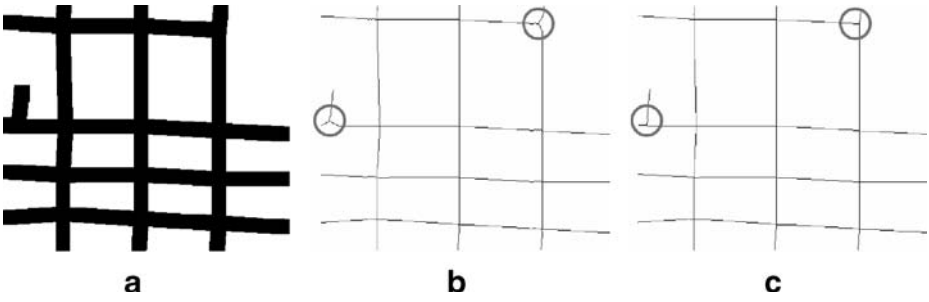

Fig. 25 The results after thinning with and without erosion. **a** After dilation. **b** Without erosion. **c** With erosion

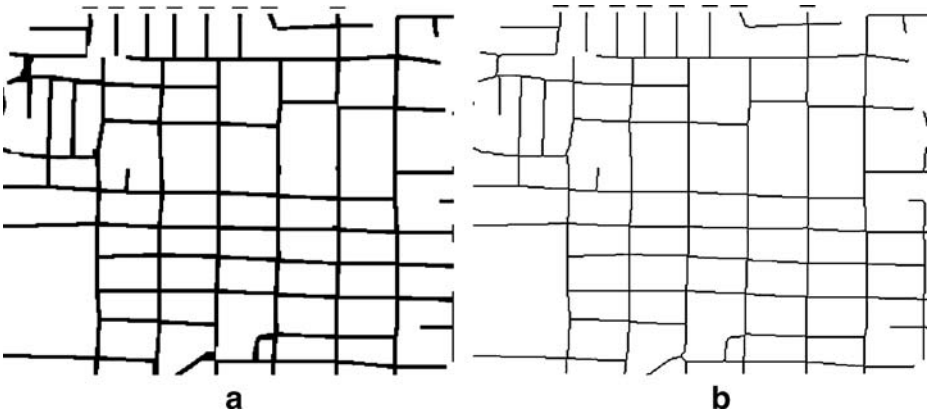

Fig. 26 TIGER/Line map before and after thinning. **a** After erosion. **b** After thinning

2.6.1 Detection of road intersection candidates

With the roads from the preprocessing steps, we need to locate possible road intersection points. A road intersection point is a point at which more than two lines meet with different tangents. To detect possible intersection points, we start by using an *interest operator*. As the name *interest operator* implies, it detects "interest points" from an image as starting points for other operators to further work on. We use the *interest operator* proposed by Shi and Tomasi [22] and implemented in OpenCV[7] to find the *interest points* as the road intersection candidates.

The *interest operator* checks the color variation around every foreground pixel to identify *interest points*, and it assigns a quality value to each interest point. If one *interest points* lies within the predefined radius R of some *interest points* with higher quality value, it will be discarded. For example, consider Fig. 27, where pixels 1 to 5 are all *interest points*. With the radius R defined as 5 pixels, salient point 2 is too close to salient point 1 which has a higher quality value. As a result, we discard salient point 2 while salient point 1 is kept as a road intersection candidate. Salient point 4 is also discarded, because it lies within the 5 pixels radius of salient point 3. Salient point 5 is considered as a road intersection point candidate, since it is not close to any other *interest points* with higher quality value. The radius R of 5 pixels is selected through experimentation. If we select a smaller radius, we will have more road intersection candidates; otherwise we will have fewer candidates. Consider the fact that road intersections are not generally near each other, the radius of 5 pixels is reasonable for processing maps. These road intersection candidates are then passed to the next module for the determination of the actual road intersections.

2.6.2 Filtering intersections, extracting road connectivity and orientation

An *interest point* could be detected on a road line where the slope of the road suddenly changes. One example is the point 5 shown in Fig. 27. So we have to filter

[7] http://sourceforge.net/projects/opencvlibrary, GoodFeaturesToTrack function.

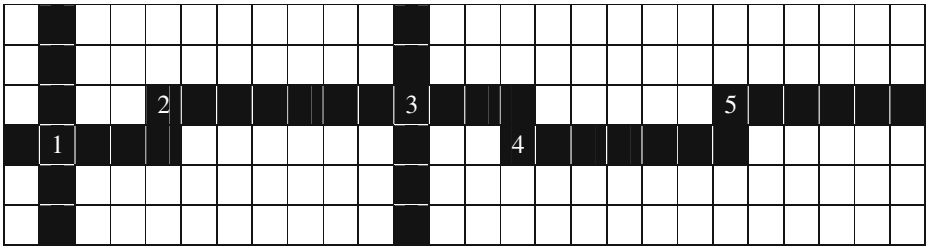

Fig. 27 The *interest points* (*black cells* are foreground pixels)

out the *interest points* that do not have the geometric property of an intersection point. Every road intersection point should have more than two line segments which meet at that point. The definition of intersection connectivity is the number of line segments intersecting at an intersection point, and it is the main criteria to filter road intersection points from *interest points*.

We assume roads on raster maps are straight within a small distance (i.e., several meters). For each of the *interest points* detected by the interest operator, we draw a rectangle around it as shown in Fig. 28. The size of the rectangle is based on the maximum length in our assumption that the road lines are straight. In our example shown in Fig. 28, we use an 11-by-11 rectangle on the raster map with resolution 2 m/pixel, which means we assume the road lines are straight within 5 pixels or 10 m (e.g., on the horizontal direction, a line of length 11 pixels is divided into 5 pixels to the left, one center pixel and 5 pixels to the right). Although the rectangle size can vary for different raster maps with various resolutions, we use a small rectangle size to assure even with lower resolution raster maps, the assumption that road lines within the rectangle are straight is still valid.

The connectivity of an *interest point* is the number of foreground pixels that intersects with this rectangle since the road lines are all single pixel width. If the connectivity is less than three, we discard the point; otherwise it is identified as a road intersection point. Subsequently, we link the *interest points* to the intersected foreground pixels on the rectangle boundaries to compute the slope (i.e., orientation) of the road lines as shown in Fig. 29.

In this step, we do not trace the pixels between the center pixel and the intersected pixels at the rectangle boundaries. In general, this step could introduce errors if the

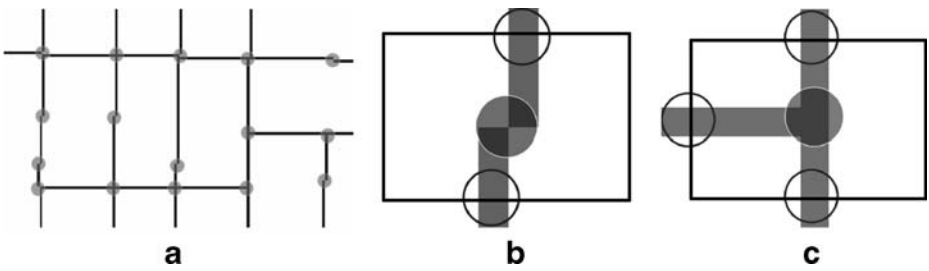

Fig. 28 The intersection candidates (*gray circles*) of a portion of the TIGER/Line Map. **a** The *interest points*. **b** Not an intersection. **c** An intersection

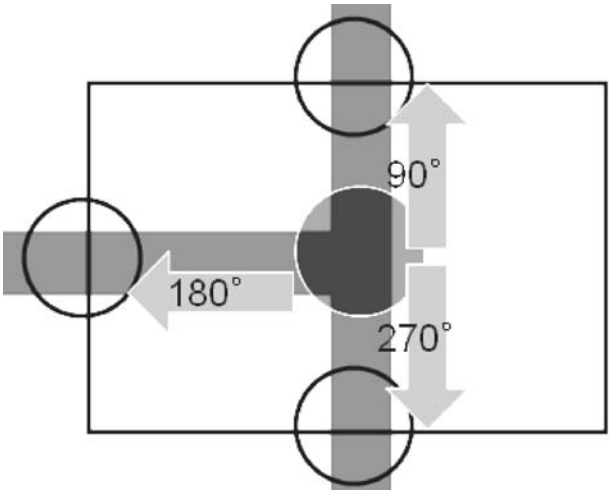

Fig. 29 Construct lines to compute orientation

intersected pixels are from other road lines which do not intersect at the center pixel or if the road lines within the rectangle are not straight. This usually happens in low-resolution maps; however, in the general case, the rectangle is much smaller than the size of a street block, and it is unlikely to have other road lines intersect or have non-straight road lines. Moreover, we save significant computation time by avoiding the tracing of every possible road line between the center and the rectangle box.

2.6.3 Localized template matching

LTM [5] is the last step in our approach, which enhances the accuracy of the extracted position, connectivity, and orientation. During the preprocessing steps to rebuild the road layer (i.e., the binary morphological operators), we might shift the road lines from their original position or even create false branches. In order to ensure accurate results, we utilize the LTM to compare the extracted results with the original raster map and return a similarity value.

For every extracted road intersection, we construct a template based on the road layer format, connectivity, and road orientation. For example, Fig. 30a shows a road intersection point with connectivity equal to 4, and the orientations of the intersected roads are 0, 90, 180, and 270 degree, respectively. If the raster map is a double-line map, we use the road width RW from the parallel-pattern tracing step with one pixel wide lines (i.e., the black lines in the figures) to construct the template as shown in Fig. 30b; otherwise, we use one pixel width lines to construct a single-line template as shown in Fig. 30c (i.e., the line width is the road width for single-line roads). After we have the template, we utilize the LTM function implemented in [5] to search locally from the position of the extracted intersection point as shown in Fig. 31. LTM will locate regions in the binary raster map that are most similar in terms of the geometry to the generated template. The outputs of LTM are the position of the matched template and a similarity value. If the similarity is larger than a pre-set threshold, we adjust the position of the intersection point; otherwise we discard the intersection point. As shown in Fig. 32, LTM adjusts the circled intersection points

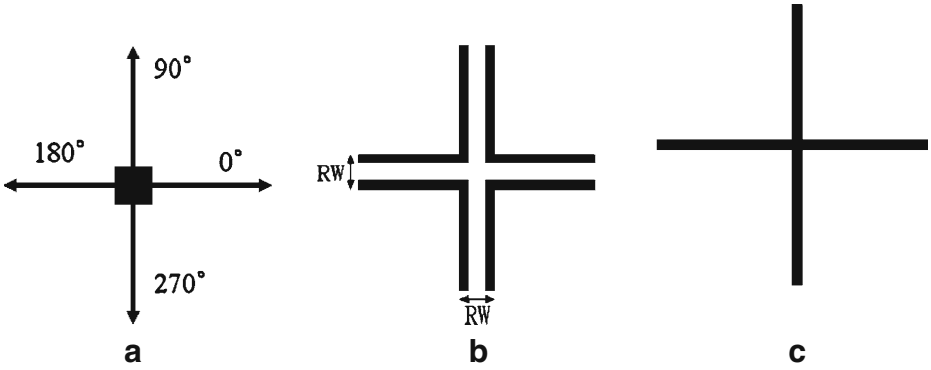

Fig. 30 Templates for different road format. **a** Extracted intersection. **b** Double-line format. **c** Single-line format

to the precise location on the original raster map, and the arrows show the directions of the adjustments. The differences are only a few pixels, so the figures need to be studied carefully to see the differences. For example, in the upper-left circle of Fig. 32a, LTM adjusts the intersection several pixels lower and makes it match the exact intersection location on the original map; for the three circles on the bottom, LTM moves the intersections to their right for exact matches.

3 Experiments

In this section, we evaluated our approach by conducting experiments on raster maps from various sources. Section 3.1 explains the test datasets and the initial parameter settings. Section 3.2 presents our evaluation methodology. Section 3.3 analyzes the experimental results and provide a comparison to our previous work [3] and an earlier paper of this work in [7]. Section 3.4 describes the execution platform and discusses the computation time.

Fig. 31 LTM (TIGER/Line Map). **a** Search within a local area. **b** Find a match

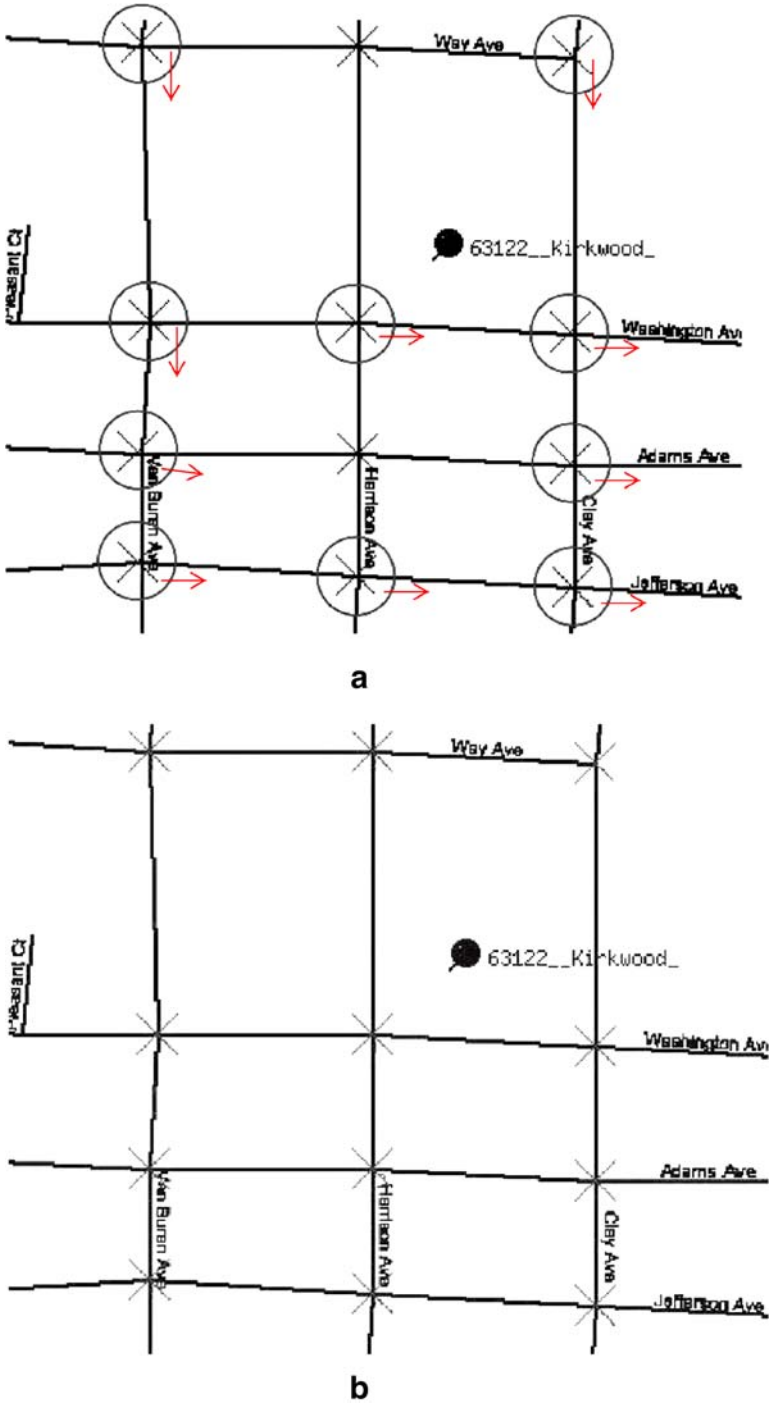

Fig. 32 Results before/after the LTM (TIGER/Line Map). **a** Before LTM. **b** After LTM

3.1 Experimental setup

We experimented with computer generated maps and scanned maps from 12 sources[8] covering cities of the United States and some European countries. We first arbitrarily selected 70 detailed street maps with a resolution range from 1.85 m/pixel to 7 m/pixel.[9] In addition, we deliberately selected 7 low-resolution abstract maps (resolutions range from 7 m/pixel to 14.5 m/pixel) to test our approach on more complex raster maps that have significant overlap between lines and characters.

We did not use any prior information about the input maps. Instead we used a set of default parameters for all the input maps based on practical experience on a small set of data (disjoint with our test dataset in this experiment), which may not produce the best results for all raster map sources but demonstrate our capability to handle a variety of maps. The size of small connected objects to be removed in the text/graphics separation step is set to 20-by-20 pixels. The number of iterations for the binary dilation and erosion operators on a single-line map are three and two respectively (i.e., a gap smaller than six pixels can be fixed). In the intersection filtering and connectivity and orientation extracting step, we used a 21-by-21 pixel rectangular box (10 pixels to the left, 10 pixels to the right plus one center pixel). The similarity threshold for the LTM is 50%. We could optimize these parameters for one particular source to produce the best results if we incorporate prior knowledge of the sources in advance.

3.2 Evaluation methodology

The output of our approach is a set of road intersection positions along with the road connectivity and the orientations. We first report the precision (correctness) and recall (completeness)[10] for the accuracy of the extracted intersection positions. For the displacement quality of the results, we randomly selected two maps from each source to examine the positional displacement. We also report the geometry similarity between the intersection templates we extracted and the original map to analyze the quality of the road connectivity and orientation.

The precision (correctness) is defined as the number of correctly extracted road intersection points divided by the number of extracted road intersection points. The recall (completeness) is defined as the number of correctly extracted road intersection points divided by the number of road intersections on the raster map. The positional displacement is the distance in pixels between the correctly extracted road intersection points and the corresponding actual road intersections. Correctly extracted road intersection points are defined as follows: if we can find a road intersection on the original raster map within a N pixel radius of the extracted road intersection point, it is considered a correctly extracted road intersection point. Based on our practical experience, if an extracted intersection is within five pixel radius to any road intersections on the original map, it usually corresponds to

[8]ESRI Map, MapQuest Map, TIGER/Line Map, Yahoo Map, A9 Map, MSN Map, Google Map, Map24 Map, ViaMichelin Map, Multimap Map, USGS Topographic Map, Thomas Brothers Map.

[9]Some of the sources do not provide resolution information.

[10]The terms precision and recall are common evaluation terminologies in information retrieval and correctness and completeness are often used alternatively in geomatics and remote sensing [12].

an actual intersections on the original map but was shifted during the extraction; otherwise it is more likely a false-positive generated during our process to rebuild the roads. Thus, N equal to five is used as the upper bound to report our results. We report the precision and recall as we vary N from 0 to 5 for a subset of testing data in Section "3.3;" and we use N equal to 5 for any other places in the paper when precision and recall are mentioned. N represents the maximum positional displacements an application can tolerate. Different usages of the extracted intersections have different tolerance levels on the value of N. For example, consider an application that utilizes the extracted intersections as seed templates to search for neighborhood road pixels on aerial imagery [13], it needs as many intersections as possible; however, it is likely to have a lower requirement on the positional accuracy of the intersections (i.e., N can be larger). On the other hand, a conflation system such as [5] requires higher positional accuracy (i.e., a smaller N) to match the set of road intersections to another set of road intersections from another source. Road intersections on the original maps are defined as the intersection points of two or more road lines for single-line maps or any pixel inside the intersection areas where two or more roads intersect for double-line maps.

The geometric similarity represents the similarity between the extracted intersection template and the binary raster map, which is computed with LTM as described in Section 2.6.3 and used in [4]. For an intersection template T with w x h pixels and the binary raster map B, the geometric similarity is defined in as:

$$GS(T) = \frac{\sum_{y=1}^{h}\sum_{x=1}^{w} T(x,y)B(X+x,Y+y)}{\sqrt{\sum_{y=1}^{h}\sum_{x=1}^{w} T(x,y)^2 \sum_{y=1}^{h}\sum_{x=1}^{w} B(X+x,Y+y)^2}} \quad (1)$$

where T(x,y) equals one, if (x,y) belongs to the intersection template; otherwise T(x,y) equals zero. B(x,y) equals one, if (x,y) is a foreground pixels on the binary raster map; otherwise B(x,y) equals zero. In other words, the geometric similarity is a normalized cross correlation between the template and the binary raster map which ranges from zero to one [4].

3.3 Experimental results

We report the results with respect to the map sources as shown in Table 2. The average precision is 95% and the recall is 75% under the set of parameters discussed in Section 3.1. In particular, for map sources like the A9 Map, the Map24 Map, and the ViaMichelin Map, the precision is 100% because of the fine quality of their maps (i.e., less noise, same width roads etc.). In our experiments the USGS Topographic Maps have the lowest precision and recall besides the low-resolution raster maps. This is because USGS Topographic Maps contain more information layers than other map sources and the quality of USGS Topographic Maps is not as good as computer generated raster maps due to the poor scan quality. The average geometric similarity of the extracted intersection template generated from LTM is 72%. We can improve the similarity if we track the line pixels when filtering the intersections to generate the templates, but it will require more computation time. Our approach generates a set of representative features for applications that require high quality of corresponding features for their matching process. In [5], a map to imagery conflation system proposed by Chen et al. build on the results of our approach (i.e., the set of

Table 2 Average precision, recall, and F-measure with respect to various sources

Map source (number of test maps)	Precision (%)	Recall (%)	F-Measure (%)
ESRI map (10)	93	71	81
MapQuest map (9)	98	66	79
TIGER/Line map (9)	97	84	90
Yahoo map (10)	95	76	84
A9 map (5)	100	93	97
MSN map (5)	97	88	92
Google map (5)	98	86	91
Map24 map (5)	100	82	90
ViaMichelin map (5)	100	98	99
Multimap map (5)	98	85	91
USGS Topographic map (10)	82	60	69
Thomas Brothers map (2)	98	65	79

road intersection templates) and achieved their goal to identify the geospatial extent of the raster map and to align the map with satellite imagery.

The value of positional displacements for two randomly selected maps from each source is 0.25 pixels on average; and the Root Mean Squared Error (RMSE) is 0.82, which means the majority of extracted intersections are within one pixel radius of the actual intersections on the map. Some extracted intersections are still not on the precise original position since we construct the LTM template using 1-pixel width roads instead of using the original road line width, which is unknown. As shown in Fig. 33, we also report the recall and precision with the positional displacement, N, varies from 0 pixels (i.e., we extract the exact position) to 5 pixels. Intuitively, the precision and recall are higher when the N increases. For applications that do not

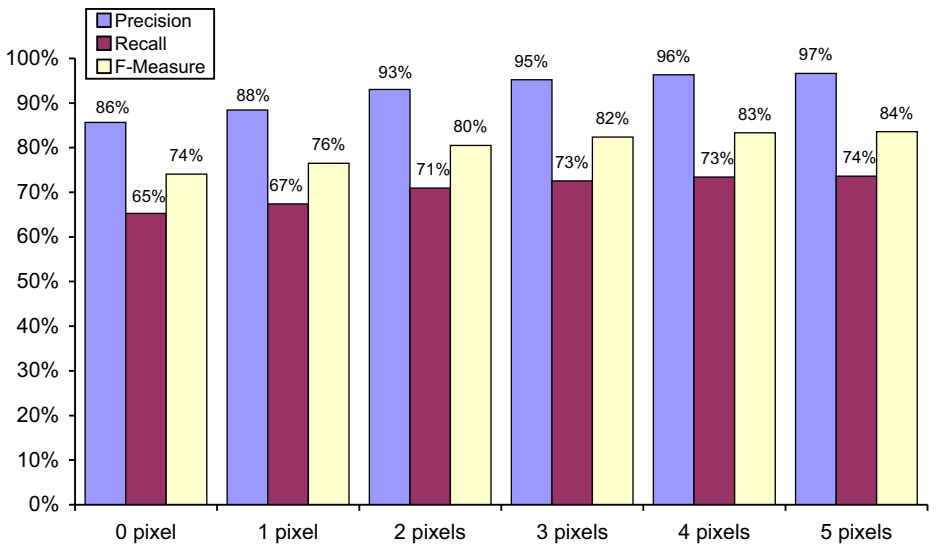

Fig. 33 The precision and recall with respect to the positional displacement

Table 3 Experimental results with respect to the resolution

	Precision (%)	Recall (%)
Resolution higher then 7 m/pixel (70 maps)	95	75
Resolution lower then 7 m/pixel (7 maps)	83	27

require finding the exact location of the intersections, we can achieve higher precision and recalls.

For comparison, we selected 7 low-resolution abstract maps (resolutions range from 7 m/pixel to 14.5 m/pixel) to test our approach on more complex raster maps that have significant overlap between lines and characters. The experimental results of the 7 low-resolution abstract maps compared to the set of 70 high-resolution maps are shown in Table 3. The low-resolution maps (i.e., resolutions lower than 7 m/pixel) have significantly lower average recall. This is because the characters and symbols touch the lines more frequently as shown in Fig. 34. During the preprocessing steps, we use a text/graphics separation program to remove the characters and labels, and it removes many of the road lines in a low-resolution map. Moreover, the size of street blocks on the low-resolution map are usually smaller than the window size we use in the intersection filter, which leads to inaccurate identification of the road orientations.

Finally, we tested our approach in this paper against an earlier paper of this work in [7] and the previous work of Chen et al. [3] using maps from the same map sources (i.e., ESRI, TIGER/Line, MapQuest, and Yahoo). The results are shown in Fig. 35. To our best knowledge, besides [7] and [3], the closest work we found is from Habib et al. [11], but they have very different assumptions of the input maps than us (they assume the maps contain only roads) and also there was no numeric results reported in the paper. We also conduct significance tests on the comparison with the precision and recall of our previous work in [7] and this paper using T-distribution at $p < 0.05$ with the 95 test maps. The precision is significantly improved from [3] and slightly improved from our earlier paper on this work [7] (the difference is statistically significant) resulting from the usage of LTM. The F-measure is also slightly improved from the previous work. For the quality of the intersection positions, we picked two

Fig. 34 A low-resolution raster map (TIGER/Line Map,7.99 m/pixel)

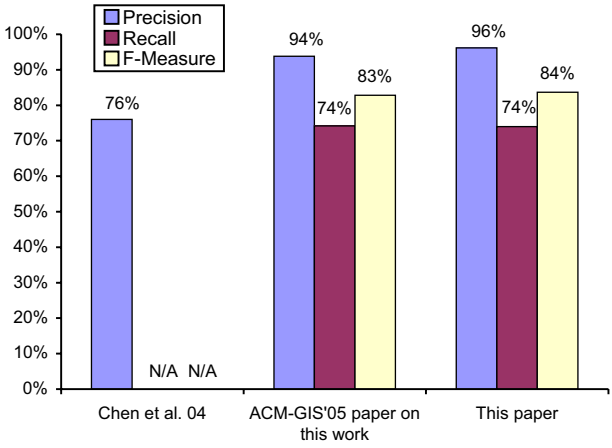

Fig. 35 Comparison with previous work

maps for each source to calculate the positional displacement. The RMSE of the positional displacement using the approach in this paper is 1.01, which is improved from 1.37 in [7] due to the use of LTM. The recall in [7] is slightly higher than the recall of this paper (the difference is not statistically significant) since the LTM filters some correct intersections with incorrect orientation and connectivity templates. Two example results from our experiments are shown in Fig. 36. In these figures, an "X" marks the one road intersection point extracted by our system.

3.4 Computation time

Our test platform is an Intel Xeon 1.8 GHz Dual Processors server with 1 GB memory and the development tool is Microsoft Visual Studio 2003. The efficiency

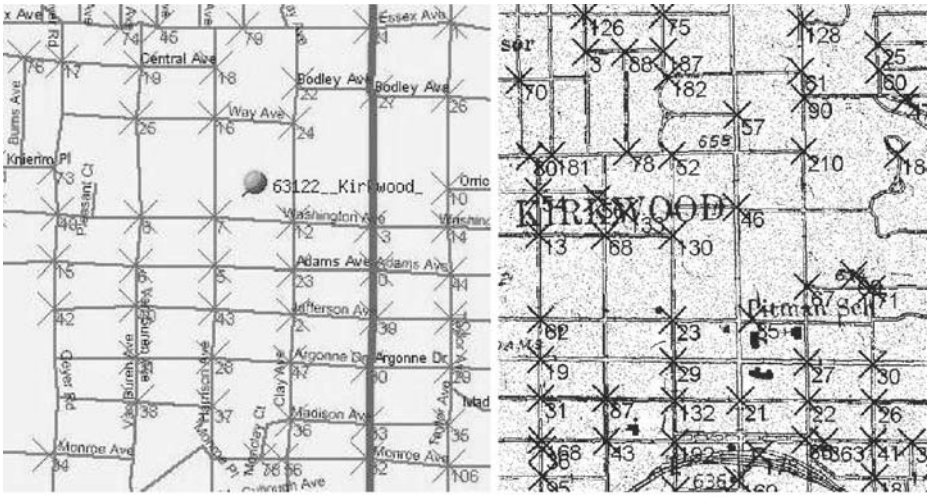

Fig. 36 Road intersection extraction. **a** TIGER/Line map. **b** USGS Topographic map

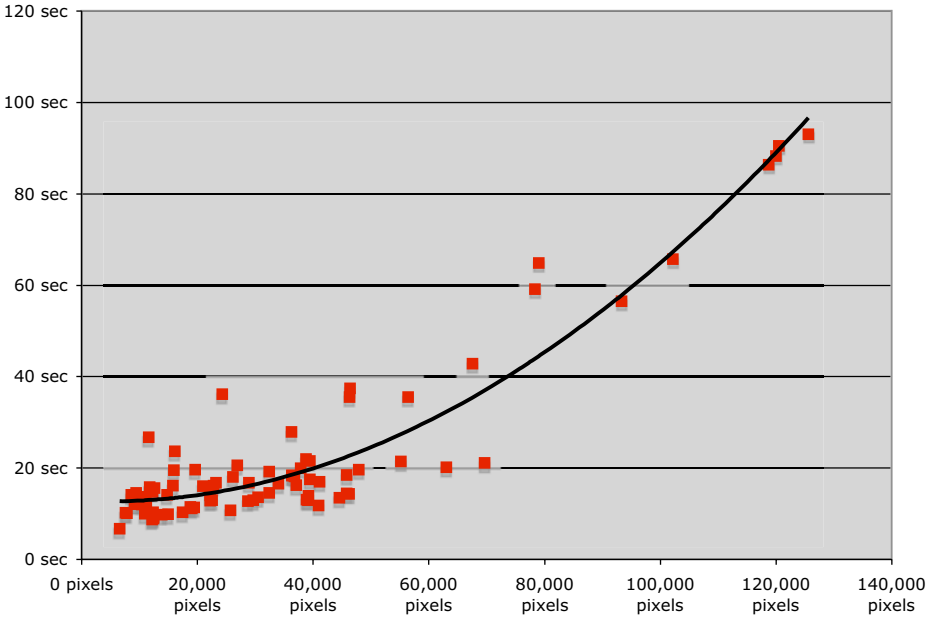

Fig. 37 Computation time is related to the number of foreground pixels

of our approach depends on the computation time for each component, and the dominant factors are the map size and the number of foreground pixels in the raster maps. As shown in Fig. 37, the computation time increases when the number of foreground pixels increases. Raster maps that contain more information need more time than others. USGS Topographic Maps are the most informative raster maps in our experiments, and it took less than one minute to extract the road intersections from an 800×600 topographic map with around 120 k foreground pixels on our testing platform. Other sources require fewer than 20 s on images of size less than 500×400. The LTM algorithm takes about 0.25 s to match an intersection point.

4 Related work

There is a variety of research on object recognition from raster maps, such as road intersection extraction [11], contour line extraction [24], general object recognition [17], [21], and text/graphics separation [2], [6], [10], [15]–[18], [25], [26].

Habib et al. [11] utilize several image processing algorithms to automatically extract road intersections from raster maps. To extract the road layer, Habib et al. simply assume there are only road lines on the input raster maps, and use an edge detector to separate the roads from the background. With the extracted road lines, Habib et al. utilize an interest operator to detect the corner points (i.e., *interest points* in our paper) on the road edges and then search for corner point groups. The centroids of the resulting groups are the road intersection points. In this case, false-positive corner points or intersections of T-shape roads significantly shift the centroids away from the correct locations. After the intersections are extracted, they extract the connectivity and orientations using manually identified knowledge of the

road format and the road width. In comparison, our approach detects the road layer format and road width automatically to rebuild the road layer. Moreover, the usage of LTM with the road width ensures the extracted intersections are on the original positions without manually verifying the results.

Salvatore and Guitton [24] use a color classification technique to extract contour lines from the topographic maps. Their technique requires prior knowledge to generate a proper set of color thresholds for a specific set of maps. However, in reality, the thresholds for different topographic maps covering different areas may vary depending on the quality of the raster map. With our approach, we separate the contour lines from the roads by distinguishing their different geometric representations. In addition, the previous work has the goal to ensure the resulting contour lines have continuity similar to the original while our focus is on the road lines that are close to the intersections.

Samet et al. [21] use the legends in a learning process to identify objects on the raster maps. Meyers et al. [17] use a verification based approach with map legends and specifications to extract data from raster maps. These approaches both need prior knowledge (i.e., legend layer and map specification) of the input raster maps, and the training process needs to be repeated when the map source changes.

Finally, much research work has been performed in the field of text/graphics separation from documents and maps [2], [6], [10], [15]–[18], [25], [26], which is related to one of our step to extract and rebuild the road layer. Among the text/graphics separation research, [2], [10], [26] assume that the line and character pixels are not overlapping and they extract characters by tracing and grouping connected objects. Cao et al. [6] detects characters from more complex documents (i.e., characters overlap lines) using the differences of the length of line segments in characters and lines. Li et al. [15, 16] and Nagy et al. [18] first separate the characters from the lines using connected component analysis and then focus on local area to rebuild the lines and characters using various methods. Generally, the text/graphics separation research emphasize on extracting characters, hence it provides a partial solution to our goal to extract the intersection templates. In Section 2.2.2, we describe how we incorporate the algorithm from [6] to remove the characters before we utilize the binary morphological operators to rebuild the roads.

The main difference between our approach and the previous work on map problems is that the previous work requires additional information hence they provide partial solutions to the problem of automatic road intersection extraction. We assume a more general scenario to handle various map sources [9]; and our approach requires no prior knowledge while it still can be tuned with additional information, if available.

5 Conclusion and future work

The main contribution of this paper is to provide a complete framework to automatically and accurately extract the intersections from raster maps. We also identify other valuable information such as the road format (i.e., single-line format or double-line format) and road width to help the extraction process.

Our approach achieves 95% precision and 75% recall on average when automatically extracting road intersections from raster maps with resolution higher than

7m/pixel without any prior information. The result is a set of accurate features that can be used to exploit other geospatial data or to intergrate a raster map with other geospatial sources, thus creating an integrated view. For example, in [5], Chen et al. used the extracted intersections to align the maps to imagery. Moreover, for a road extraction application, the georeferenced road intersections can be used as seed templates to extract the roads from imagery [13]. In the work in [9], Desai et al. applied our automatic intersection extraction technique on maps returned from image search engines and successfully identify the road intersection points for geospatial fusion systems to identify the geocoordinates of the input maps.

There are three primary assumptions of our current approach. First, the background pixels must be separable using the difference in luminosity level from the foreground pixels. This means that the background pixels must have the dominant color in the raster maps. On certain raster maps that contain numerous objects and the number of foreground pixels is larger than that of the background pixels, those foreground objects seriously overlap with each other making the automatic processing nearly impossible. Even if we can remove the background pixels on these raster maps, removing noisy objects touching road lines results in too many broken road segments that are difficult to reconnect. Second, although our approach works with no prior knowledge of the map scales, low-resolution raster maps (i.e., in our experiments, above 7 m/pixel) may result in low precision and recall. Third, as mentioned in Section 2.2.2, if the characters on the input map are significantly different than our preset value, we cannot remove the character from lines and the results will have many incorrectly identified intersections.

In future work, we plan to address these issues. First, we plan to exploit texture classification methods [20] to handle those raster maps in which the background color is not the dominate color such as tourist maps. Second, although the text/graphics separation program performs well with one set of default parameters in our experiments, we still need to specify these parameters for each map to achieve the best results. Instead of tuning for every map, we want to utilize classification methods in the frequency domain [8], [14] to separate line and character textures, which do not require any geometric parameters. Third, we want to further enhance the extracted road layer. In our approach, we do not focus on repairing the road network, but rather on rebuilding the roads close to the intersections. Hence we generate a comparatively coarse road layer from the original raster map. With the help of vectorization algorithms, we can further repair the road layer and generate the road vector data from the raster maps.

Acknowledgements This research is based upon work supported in part by the United States Air Force under contract number FA9550-08-C-0010, in part by the National Science Foundation under Award No. IIS-0324955, in part by the Air Force Office of Scientific Research under grant number FA9550-07-1-0416, in part by a gift from Microsoft, and in part by the Department of Homeland Security under ONR grant number N00014-07-1-0149. The U.S. Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them. We would like to thank Dr. Chew Lim Tan for his generous sharing of their code in [6].

References

1. Agam G, Dinstein I (1996) Generalized morphological operators applied to map-analysis. In: The proceedings of the 6th international workshop on advances in structural and syntactical pattern recognition. Springer-Verlag, pp 60–69
2. Bixler JP (2000) Tracking text in mixed-mode documents. In: The ACM conference on document processing systems, ACM, Santa Fe, New Mexico, pp 177–185
3. Chen C-C, Knoblock CA, Shahabi C, Chiang Y-Y, Thakkar S (2004) Automatically and accurately conflating orthoimagery and street maps. In: The 12th ACM international symposium on advances in geographic information systems, ACM, Washington, D.C., pp 47–56
4. Chen C-C, Knoblock CA, Shahabi C (2006) Automatically conflating road vector data with orthoimagery. Geoinformatica 10(4):495–530
5. Chen C-C, Knoblock CA, Shahabi C (2008) Automatically and accurately conflating raster maps with orthoimagery. GeoInformatica (in press)
6. Cao R, Tan CL (2001) Text/graphics separation in maps. In: The 4th international workshop on graphics recognition algorithms and applications, Kingston, Ontario. Springer Verlag, pp 167–177
7. Chiang Y-Y, Knoblock CA, Chen C-C (2005) Automatic extraction of road intersections from raster maps. In: The 13th ACM international symposium on advances in geographic information systems, Bremen, ACM
8. Chiang Y-Y, Knoblock CA (2006) Classification of line and character pixels on raster maps using discrete cosine transformation coefficients and support vector machines. In: The international conference on pattern recognition, Hong Kong, IEEE Computer Society
9. Desai S, Knoblock CA, Chiang Y-Y, Desai K, Chen C-C (2005) Automatically identifying and georeferencing street maps on the web. In: The 2nd international workshop on geographic information retrieval, Bremen, ACM
10. 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
11. Habib AF, Uebbing RE (1999) Automatic extraction of primitives for conflation of raster maps. Technical report. The Center for Mapping, The Ohio State University
12. Heipk C, May H, Wiedemann C, Jamet O. (1997) Evaluation of automatic road extraction. In: The ISPRS Conference, vol 32, pp 3–2W3
13. Koutaki G, Uchimura K (2004) Automatic road extraction based on cross detection in suburb. In: Image processing: algorithms and systems III. Proceedings of the SPIE, vol 5299, pp 337–344
14. Keslassy I, Kalman M, Wang D, Girod B (2001) Classification of compound images based on transform coefficient likelihood. In: The international conference on image processing, Thessaloniki, IEEE
15. Li L, Nagy G, Samal A, Seth S, Xu Y (1999) Cooperative text and line-art extraction from a topographic map. In: Proceedings of the 5th international conference on document analysis and recognition, IEEE
16. Li L, Nagy G, Samal A, Seth S, Xu Y (2000) Integrated text and line-art extraction from a topographic map. IJDAR 2(4):177–185
17. 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
18. Nagy G, Sama A, Set S, Fisher T, Guthmann E, Kalafala K, Li L, Sivasubraniam S, Xu Y (1997) Reading street names from maps - Technical challenges. In: Procs. GIS/LIS conference, pp 89–97
19. Pratt WK (2001) Digital image processing: PIKS inside, 3rd edn. Wiley-Interscience, New York
20. Randen T, Husoy JH (1999) Filtering for texture classification: a comparative study. IEEE Trans Pattern Anal Mach Intell 21(4):291–310
21. Samet H, Soffer A (1994) A legend-driven geographic symbol recognition system. In: The 12th international conference on pattern recognition, IEEE, Jerusalem, vol 2, pp 350–355, October
22. Shi J, Tomasi C (1994) Good features to track. In: The IEEE conference on computer vision and pattern recognition, IEEE, Seattle
23. Sezgin M, Sankur B (2004) Survey over image thresholding techniques and quantitative performance evaluation. J Electron Imaging 13(1):146–165
24. Salvatore S, Guitton P (2001) Contour line recognition from scanned topographic maps. Technical report. University of Erlangen

25. Tang YY, Lee S-W, Suen CY (1996) Automatic document processing: a survey. Pattern Recogn 29(12):1931–1952
26. Velázquez A, Levachkine S (2003) Text/graphics separation and recognition in raster-scanned color cartographic maps. In: The 5th international workshop on graphics recognition algorithms and applications, Barcelona, Catalonia
27. Zack GW, Rogers WE, Latt SA (1977) Automatic measurement of sister chromatid exchange frequency. J Histochem Cytochem 25(7):741–753

Yao-Yi Chiang is currently a Ph.D. student at the University of Southern California (USC). He received his B.S. in Information Management from National Taiwan University in 2000 and then his M.S. degree in Computer Science from the USC in December 2004. His research interests are on the automatic fusion of geographical data. He has worked extensively on the problem of automatically utilize raster maps for understanding other geospatial sources and has wrote and co-authored several papers on automatically fusing map and imagery as well as automatic map processing. Prior to his doctoral study at USC, Yao-Yi worked as a Research Scientist for Information Sciences Institute and Geosemble Technologies.

Craig A. Knoblock is a Senior Project Leader at the Information Sciences Institute and a Research Professor in Computer Science at the USC. He is also the Chief Scientist for Geosemble Technologies, which is a USC spinoff company that is commercializing work on geospatial integration. He received his Ph.D. in Computer Science from Carnegie Mellon. His current research interests include information integration, automated planning, machine learning, and constraint reasoning and the application of these techniques to geospatial data integration. He is a Fellow of the American Association of Artificial Intelligence.

Cyrus Shahabi is currently an Associate Professor and the Director of the Information Laboratory (InfoLAB) at the Computer Science Department and also a Research Area Director at the NSF's Integrated Media Systems Center at the USC. He received his B.S. in Computer Engineering from Sharif University of Technology in 1989 and then his M.S. and Ph.D. degrees in Computer Science from the USC in May 1993 and August 1996, respectively. He has two books and more than hundred articles, book chapters, and conference papers in the areas of databases, geographic information system (GIS) and multimedia. Dr. Shahabi's current research interests include Geospatial and Multidimensional Data Analysis, Peer-to-Peer Systems and Streaming Architectures. He is currently an associate editor of the IEEE Transactions on Parallel and Distributed Systems and on the editorial board of ACM Computers in Entertainment magazine. He is also a member of the steering committees of IEEE NetDB and the general co-chair of ACM GIS 2007. He serves on many conference program committees such as VLDB 2008, ACM SIGKDD 2006 to 2008, IEEE ICDE 2006 and 2008, SSTD 2005 and ACM SIGMOD 2004. Dr. Shahabi is the recipient of the 2002 National Science Foundation CAREER Award and 2003 Presidential Early Career Awards for Scientists and Engineers. In 2001, he also received an award from the Okawa Foundations.

Ching-Chien Chen is the Director of Research and Development at Geosemble Technologies. He received his Ph.D. degree in Computer Science from the USC for a dissertation that presented novel approaches to automatically align road vector data, street maps and orthoimagery. His research interests are on the fusion of geographical data, such as imagery, vector data and raster maps with open source data. His current research activities include the automatic conflation of geospatial data, automatic processing of raster maps and design of GML-enabled and GIS-related web services. Dr. Chen has a number of publications on the topic of automatic conflation of geospatial data sources.