Digital Image Segmentation for Object-Oriented Halftoning

Zuguang Xiao [a], Mengqi Gao [a], Lu Wang [a,1], Brent Bradburn [b], Jan Allebach [a]
[a] School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47906, U.S.A.
[b] Hewlett-Packard Co., Boise, ID 83714, U.S.A.

Abstract

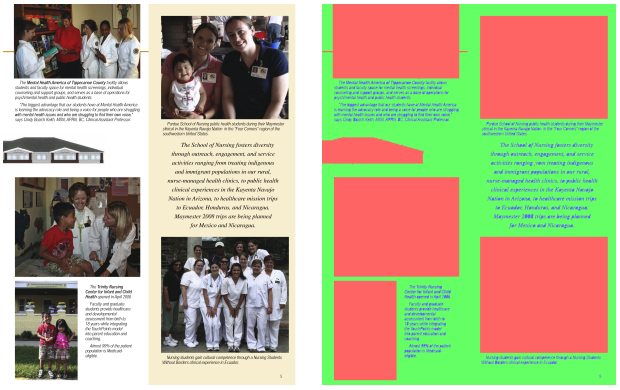
The electrophotographic (EP) process, which is widely used in imaging systems such as laser printers, is susceptible to printing artifacts if we render the smooth areas of the images with high frequency halftone screens. However, applying low frequency halftone screens over the whole page will restrict the ability to render the fine details [1]. The solution proposed by Park [2] et al is to apply different frequency screens to different parts of the page – also referred to as object-oriented halftoning. But it requires a correct object map to be generated. With miscellaneous segmented objects in a given image, an object map will classify all the image objects into three categories: raster (pictures or photos), vector (background and gradient) and symbol (symbols and texts), with raster and symbol objects considered as high frequency objects and vector objects as low frequency objects. An overall improvement of the print quality can be achieved if symbol and raster objects are rendered with high frequency, and vector objects with low frequency. Although the object map can be extracted from the page description language (PDL) directly, some components may not be correctly classified [1]. To obtain a correct object map from the page image, not the PDL, an object map generating algorithm is proposed in this paper. The algorithm uses a strip-based processing, which only requires a strip of the image to be buffered. And it is very memory efficient, making it ideal for hardware implementation.

1.0 Introduction

An object map is a matrix of labels, indicating what type of object each pixel belongs to. Figure 1(b) shows an example of the object map of the input image Figure 1(a). Three types of object are represented by different color codes in Figure 1(b): red for raster objects, blue for symbol objects, and green for vector objects. Different objects in an input image have different properties: A symbol object is usually small, and has sharp edges and a smooth interior. A vector object is usually large, and it is smooth. Raster objects can be either large or small, and they are always very rough. Only the two features – the size of an object and the roughness of an object are needed to classify a component. To classify all image objects into symbol, raster, and vector objects, if we can identify symbol and vector objects, the remaining unclassified objects will be raster objects. Symbol objects can be partitioned into symbol edge objects and symbol interior objects. So three binary images – one to find the symbol edge objects, one to find the symbol interior objects, and one to find the vector objects, are generated for connected component analysis.

To be suitable for hardware implementation, the choice of the connected component algorithm requires: a small number of passes, no random memory access, and minimal complexity and memory usage. The classic two-pass connected component algorithm [3], which is a two-pass raster order scanning process, satisfies all these requirements except for its tremendous memory consumption. The amount of memory depends on the complexity of the image, but is limited to the size of the image. However, if we only process a strip of the image at a time with the classic algorithm, the amount of memory will be reduced to the size of a strip, which is a big savings. But the discontinuities between two strips need to be taken care of. To further push down the memory consumption, a label recycling mechanism will be introduced. In addition, we use a union-find data structure with path compression to ensure fast memory access and efficiency for resolving label equivalence.

This paper[2] is organized as follows: In Section 2.1, we will first talk about how to generate the three binary images for connected component analysis. Section 2.2 will introduce the classic two-pass connected component algorithm; and its data structure will be described in Section 2.3. Section 2.4 will explain how to adapt the classic algorithm for strip-based processing and how to achieve memory efficiency. The top-level hardware architecture for our algorithm is described in Section 2.5. At last, we will provide some results of the algorithm and a performance analysis in Section 3.



(a) Input Image. (b) Object Map (red for raster; blue for symbol; green for vector).

Figure 1: An input image and its object map.

2.0 Methods

2.1 Generating binary images

To segment a color image, a common method is to first apply an edge detection filter, followed by thresholding. We use a Sobel

[1] Now with Google Inc., Mountain View, CA 94035

[2] Research supported by HP Inc., Boise, ID 83714.

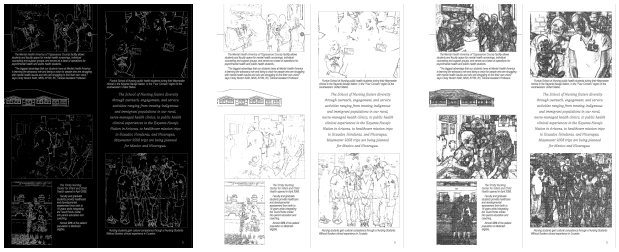
detection filter because it is simple, and easy for implementing hardware speed acceleration [4]. It contains two 3×3 mask windows, one to detect horizontal gradients and one to detect vertical gradients:

$$Gx = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad Gy = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The edge magnitude (EM) is defined as:

$$EM = \frac{1}{3} \sum_{i=r,g,b} \sqrt{(Gx * image[i])^2 + (Gy * image[i])^2} \qquad (1)$$

If we threshold the EM (image) from the Sobel operation with two threshold values, Ts_edge (strong edge threshold) and Tw_edge (weak edge threshold), we will be able to generate the following three binary images in Figure 2 from the input image Figure 1(a).



(a) Strong Edge Map. ($EM \geq Ts\_edge$) (b) Non-Strong Edge Map. ($EM < Ts\_edge$) (c) Non Edge Map. ($EM < Tw\_edge$)

Figure 2: Three binary images from Sobel operation.

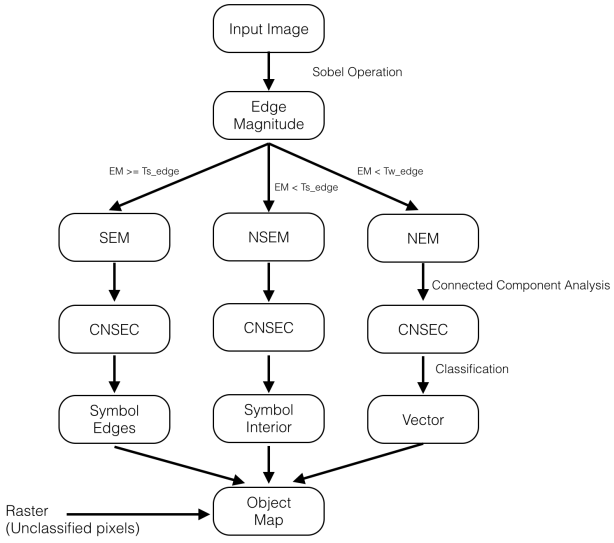
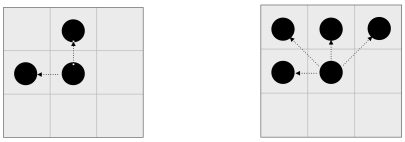


Figure 3: Overall algorithm structure.

The pixels we are interested in in each binary image are the white pixels whose EM satisfy the threshold condition. By taking a closer look at each binary image, we can see that in Figure 2(a) all the white pixels are actually the edge pixels of symbol objects and raster objects. If we could remove all the raster object pixels, we will have only symbol edge pixels retained in the first binary image. For the second image Figure 2(b), the white pixels are the interior pixels of symbol objects, as well as raster and vector objects. The objects that we want to find from this binary image are symbol interior objects. At last, the white pixels in Figure 2(c) are also interior pixels; and our interest is to find vector objects only. If we have only symbol edge pixels in the first binary image, symbol interior pixels in the second binary image, and vector pixels in the third binary image, then the remaining unclassified pixels will be raster. By using unique labels for each type of the object, we will have the final object map. However, this requires connected component analysis to extract the features for classification, and a labelling process to generate those unique labels. Figure 3 illustrates the overall algorithm structure. Connected strong edge components (CSEC), Connected non-strong edge components (CNSEC) and Connected non-edge components (CNEC) are the connected set of the white pixels in each binary image: SEM, NSEM, and NEM, respectively.

2.2 Classic CCL Algorithm

The connected component Labelling (CCL) algorithm is a process that uniquely assign labels to each connected set of foreground pixels in a binary image. The definition of connectivity is shown in Figure 4. For 4-connectivity, which is in Figure 4(a),



(a) 4-connectivity. (b) 8-connectivity.

Figure 4: Neighborhood Connectivity Context.

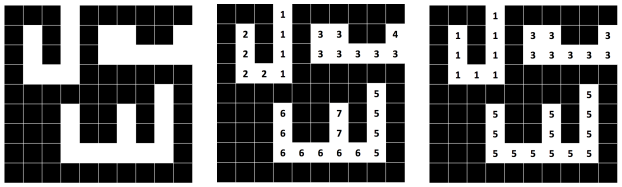
if a foreground pixel is the Northern and Western neighbor of the centered foreground pixel, then these two pixels are considered as connected. Figure 4(b) shows an 8-connectivity neighborhood context, where besides the Northern or Western neighbors, the Northwestern and Northeastern locations are also considered as connected neighbors. For our application, 4-connectivity is used because it is less complex; and there is a significant difference in the performance compared to 8-connectivity. So 4-connectivity will be used for the remainder of the paper without further mention. Each connected set of pixels is also referred as a connected component. Sometimes, the labelling process is accompanied by feature extraction, if we are interested in the properties of each component. The classic two-pass connected component [5] that will be incorporated into our algorithm is described in Figure 5.

Figure 6 shows an example of a binary image after being processed by the classic CCL algorithm. Figure 6(a) is the input binary image with white pixels representing the foreground pixels. Figure 6(b) contains the labels assigned after the first-pass. During the first-pass, we are also extracting features and recording the equivalence if there are different labels in the neighborhood context.

After the second-pass, as shown in Figure 6(c), each connected region only has one unique label, which is the smallest label among all the equivalent labels. However, for our application, we are interested in finding only one type of object for each binary image (symbol edges object for SEM, symbol interior objects for NSEM, and vector objects for NEM). So we only need

1. First Pass - Iterate pixel by pixel in raster order:
if *The pixel is foreground (white pixel)* **then**
  Get the neighborhood context of the current pixel
  if *There are no neighbors*— **then**
    uniquely assign the current pixel and continue
  else
    Assign the smallest label of the neighborhood and store the equivalence
  end
else
  Skip the current pixel
end
2. Second Pass - Iterate pixel by pixel in raster order:
if *The pixel is foreground (white pixel)* **then**
  Relabel the pixel with the lowest equivalence label
else
  Skip the current pixel
end

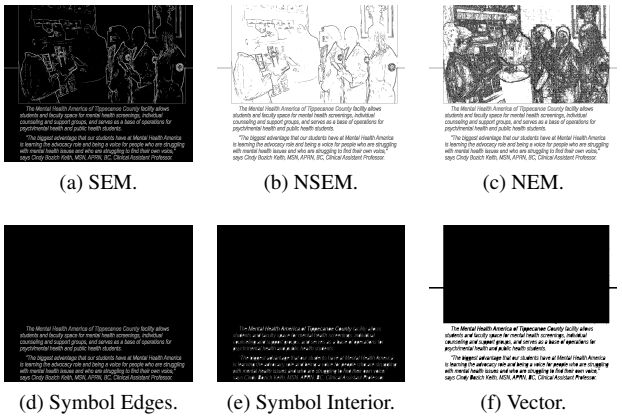
Figure 5: Algorithm 1 – Classic CCL.



(a) Original binary image. (b) First-Pass Labelling. (c) Second-Pass relabelling.

Figure 6: Classic CCL algorithm.

two types of labels after the second-pass: Class 1 (C1) represents a pixel that belongs to the type of the object that we are trying to find; and Class 2 (C2) does not. As we can see from the example provided in Figure 7, after the second-pass through the original binary images in Figures 7(a) - (c), the relabelled images are also binary images but only the pixels of interest to us remain. These pixels are Class 1 pixels, represented by the white color in Figure 7(d) - (f). The classification of pixels will be discussed later.



(a) SEM. (b) NSEM. (c) NEM.

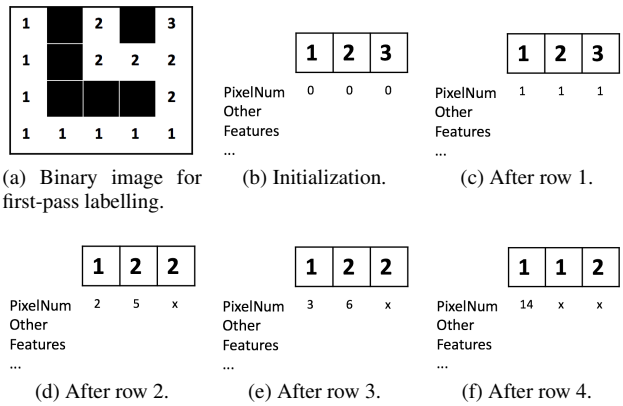
(d) Symbol Edges. (e) Symbol Interior. (f) Vector.

Figure 7: Images from edge detection, and CCL. (a) - (c) are the three input binary images; (d) - (f) are the relabelled binary images after the second-pass of CCL.

2.3 Data Structure for CCL Algorithm

The data structure we use is a 1-D array that is similar to the union-find data structure [6]. The 1-D array, referred to as the component array in this paper, is initialized with its own index; and each array element is also mapped to a set of features by a pointer. During the first-pass labelling, when a new label is needed, we will scan through the component array from the smallest index until we find an available element. Then, its index will be used as the new label and a corresponding feature set will be created, after which this element will be marked as unavailable. For every white pixel encountered during the first-pass scan, its corresponding feature set in the component array will be updated.

In the situation that an equivalence needs to be recorded, it is achieved by first finding the root of these two indices, then modifying the content indexed by the larger root to the index of the smaller root. At the same time, we merge the feature set of the larger root to the smaller root. Figure 8 illustrates how the merging algorithm works. Whenever a component is merged to its root component, its feature set will not be needed since the root component now contains all the combined features. Hence the memory for that feature set can be removed from the heap. However, the equivalence relation should always hold; and it will be optimized by path compression to reduce the traverse distance from a component to its root. For the future labelling of a pixel, instead of directly taking the label from its neighbors, we will use the root label of the neighbors; and only the feature sets of the root components will be updated.



(a) Binary image for first-pass labelling. (b) Initialization. (c) After row 1.

(d) After row 2. (e) After row 3. (f) After row 4.

Figure 8: Illustration of how the component array evolves as we do first-pass labelling row by row.

After the first-pass labelling of the whole page, we will classify all the root components in the component array before we start the second-pass. The classification result is binary-valued, either Class 1 or Class 2; and it is recorded in the feature set of the root component. All the other components rooted at that component will carry the same classification result. The classification of all the root components can be done by a one-pass scan of the component array, which has approximately linear time complexity [7]. The algorithm is described in Figure 9. When all the root components are classified, we start the second-pass, which is described in Figure 10.

The main drawback of the classic CCL algorithm is its large memory consumption. During the first-pass, the number of unique labels will be determined by the number of unmerged components in the whole page. For example, for the binary im-

```
for Each component in the component array from the
largest index do
    root component ← findRoot (current component);
    if The root component has been classified then
        continue;
    else
        classify the root component and record the
        classification result;
    end
end
```

Figure 9: Algorithm 2 – Classification of component array.

age in Figure 6(a), where there are only three disjoint regions, seven labels are used during the first-pass. The number of labels will then determine the length of the component array. Buffering all the labels and the whole component array until the end of the page during the first-pass makes the algorithm impossible for hardware implementation. However, if we look at the fifth row in Figure 6(b), where the first white pixel is labelled with "5", components 1-4 have ended already. Instead of continuing down to assign labels, if we could classify those ended components and relabel those previous pixels, then their labels would not be needed anymore and could be recycled for the following pixels to use. What's more, the memory for their feature sets could also be released. Next, we will propose a strip-based processing method that will be delineated in the following section.

```
Iterate pixel by pixel in raster order;
L ← read the label of this pixel;
if L ≠ 0 then
    Comp_L ← Component Array [L];
    Comp_Root ← findRoot(Comp_L);
    CL ← read classification result of Comp_Root;
    Replace label L with CL;
else
    Skip the current pixel;
end
```

Figure 10: Algorithm 3 – Second-pass of CCL.

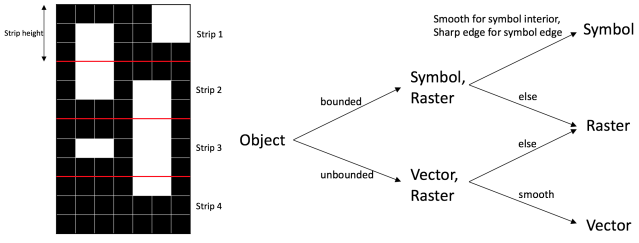
2.4 Strip-based CCL Algorithm

If we define a strip to be a little bit taller than the height of a regular text character and the same width as the input image, then an input image can be divided into many strip regions, as shown in Figure 11(a). The red lines are the strip boundaries that separate every pair of adjacent strips. Now, an object in the binary image can fall into one or several strip regions. We can group all the objects by the number of strips they span. If a component crosses fewer than two strip boundaries, it will be defined as a bounded component, otherwise it is an unbounded component, like the one at the bottom in Figure 11(a). For the three object types we have: symbol, raster, and vector, a symbol object is usually small. So it is expected to be a bounded component; and a vector object which is usually very large is expected to be an unbounded component. The raster objects can be either bounded or unbounded, but can be distinguished by their roughness. The classification is summarized in Figure 11(b).

For our first-pass labelling, it is the same as before, except that we only label one strip at a time. But at the start, we label the first two strips. Then, we classify all the root components that are above the current strip boundary. Figure 12(a) indicates that the first two strips have been labeled, and only component 1 and component 2 have been classified. Assuming component 1 is classified as Class 1 (a component of interest) and component 2 as Class 2 (not of interest), their classifications are recorded in each root component in the component array. The second-pass will start right after the classification is done; and only the previous strip is processed, which is the first strip in Figure 12(b). The reason why we do not process the second strip is that component 3 has not yet been classified. Figure 12(b) is the result after the second-pass of the first strip. As we can see, all the labels in the first strip have been replaced by their class labels. Also, if a component ended in the previous strip, for example component 1 in Figure 12(b), its memory will be recycled; and its location in the component array will be marked as available. The second-pass will be followed by the first-pass labelling of the next strip. In Figure 12(c), the label "1" recycled from the previous component is now used for this new component. The first-pass labelling of the current strip and the second-pass relabelling of the previous strip will alternate downward with a classification step between each first-pass labeling and second-pass labeling until the whole page has been relabeled.

There is a special case; and that is component 3, which is an unbounded component. For an unbounded component, we will force it to be classified at the second strip boundary it crosses, as shown in Figure 12(c). The classification is based on the features of the pixels that are above its second boundary. In this example, we do not take the bottom two pixels in Figure 12(c) into consideration when we classify component 3. Also, when we label strip 4 in Figure 12(e), instead of assigning label "3" to the to bottom two pixels, we will assign the class label directly, since component 3 has already been classified. This allows the class labels to carry across the strip boundaries, and prevents label "3" from propagating all the way down the page, and being unavailable for recycling.

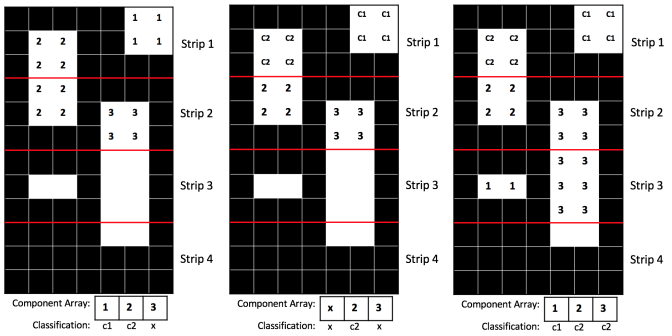
Even if we recycle components at each strip boundary, there are still a lot of components, such as single pixel components that are not of interest to us. However, these still need the memory maintained before we reach the strip boundary. An extra function is added to our algorithm that will check at each row all the root components to determine which of them have already ended. This can be done by comparing the maximum vertical coordinate of a root component to the row counter, which is pointing to the current row to which we are assigning labels. If a root component has ended on the current row, we could classify it right away. And



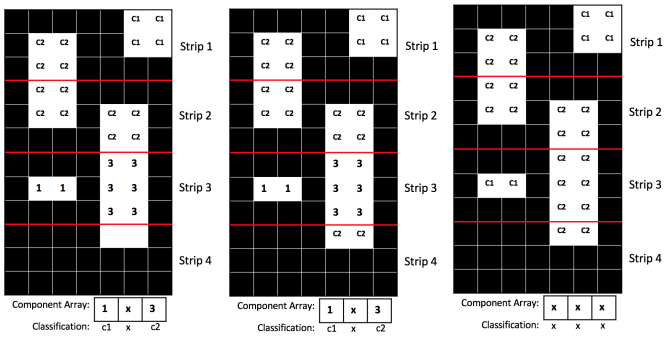
(a) Binary image separated into strips.

(b) Classification of objects.

Figure 11: Illustration of strip-based processing. (a) is the binary image with strip boundaries. (b) is the classification algorithm.



(a) First-pass on strip 1&2. (b) Second-pass on strip 1. (c) First-pass on strip 3.



(d) Second-pass on strip 2. (e) First-pass on strip 4. (f) Second-pass on strip 3.

Figure 12: Strip-based processing.

if the classification result is Class 2, we will free and recycle this root component, as well as all its leaf components; but the labels (the same as the indices of their associated components) remain unchanged. This allows us to recycle those unneeded components more frequently. However, during the relabelling process of the second-pass, it is possible that a pixel has no component associated with it because its associated component may have been freed. If this is the case, this pixel will be classified as Class 2, because only a Class 2 component will be freed during the first-pass. If a pixel does have a component associated with it, it does not imply that this pixel is a member of this componenft. We will have to compare the location of this pixel to the vertical range of this component. If the pixel is outside of this range, it implies that this pixel is not a member; and thus it should be classified as Class 2. Otherwise, the pixel is a member of this component; and it should carry the same classification result as the component. An example of this algorithm is provided in Figure 13.

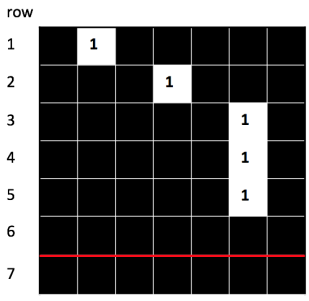


Figure 13: Recycling of labels.

As illustrated in Figure 13, after we have assigned labels to the first row, the first pixel is assigned label 1; and it has ended on the first row. We classify it and assume it is Class 2. Then its component and label will be recycled immediately. The recycled label will be used for the next pixel on row two. Assume it is also Class 2, then the label will be recycled again for the next component to use. Now if the last component has been classified as Class 1, its component (component 1) and label (label "1") will be retained; and label "2" will become the next label to use. During the second-pass, the top two pixels that have label "1" are clearly not the members of component 1 (the bottom three white pixels are). However, these two pixels are not in the vertical range of component 1, which spans from row 3 to row 5. Hence those two pixels will be relabeled with Class 2 directly. In contrast, those three adjacent pixels which are in the range will be relabeled with Class 1, which is the same class that is assigned to component 1.

2.5 Basic Architecture

The architecture of our algorithm is shown in Figure 14. Row Buffer has three rows of input image pixels. At each processing cycle, only one row from the input image will be loaded to replace the oldest row in Row Buffer. Once Row Buffer is reloaded, it will be processed by Edge Detection Logic which will generate a row of Edge Magnitude Data. The CCL logic will first threshold the Edge Magnitude data, then perform first-pass labelling of connected component algorithm on them. The communication between CCL Logic and Label Buffer is bidirectional, because the labels in Label Buffer are the labels of the previous row, which will be used as we are assigning labels to the current row. And the labels assigned for the current row will be overwritten back to Label Buffer. As we are assigning labels, the features of each pixel are extracted to Component Array; and the equivalent components are merged at any time when there is conflict in the neighborhood context. After we have assigned a row of labels into Label Buffer, those labels will be copied to the corresponding row in Object Map. The above process is repeated until the set of rows

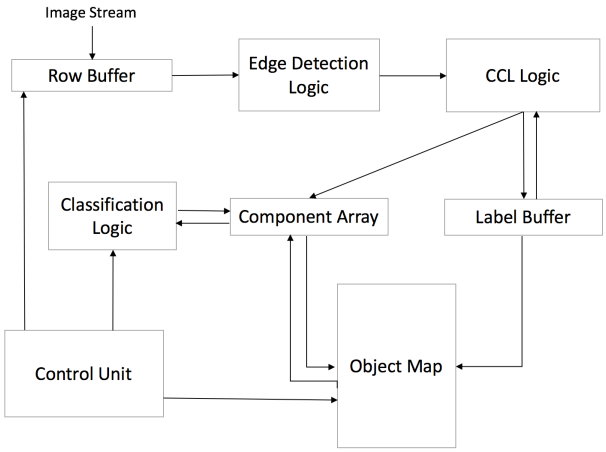


Figure 14: Basic architecture for strip-based algorithm.

in Object Map reaches a strip boundary. Row Buffer will stop loading input pixels, when signaled by Control Unit. The latter will command Classification Logic to classify all the components in Component Array, then record the classification results back to Component Array. After classification is done, Control Unit

will send a signal to Object Map to read the classification results from Component Array to replace the labels of the previous strip, which is the second-pass of the connected component algorithm. This completes the processing of one strip. The remaining strips will be processed in the same way.

3.0 Experimental Results

The test pages in Figures 15(a)-(b) have 6400×4928 pixels, at 600 ppi. Figures 15(b)-(c) are the corresponding object maps. The strip is set to a height of 80 pixels. For each object map, the symbol objects are colored with blue, raster objects with red, and vector objects with green. However, the symbol "1956" in Figure 15(b) is colored with green in Figure 15(d). This misclassification is caused by the fact that the symbol is an unbounded component. Also, its interior is very smooth. But for our application, which is object-oriented halftoning, this misclassification is acceptable since if a symbol object is too big, it will behave like a vector object: large and smooth. And it is more reasonable to render those large smooth objects with low frequency screening.

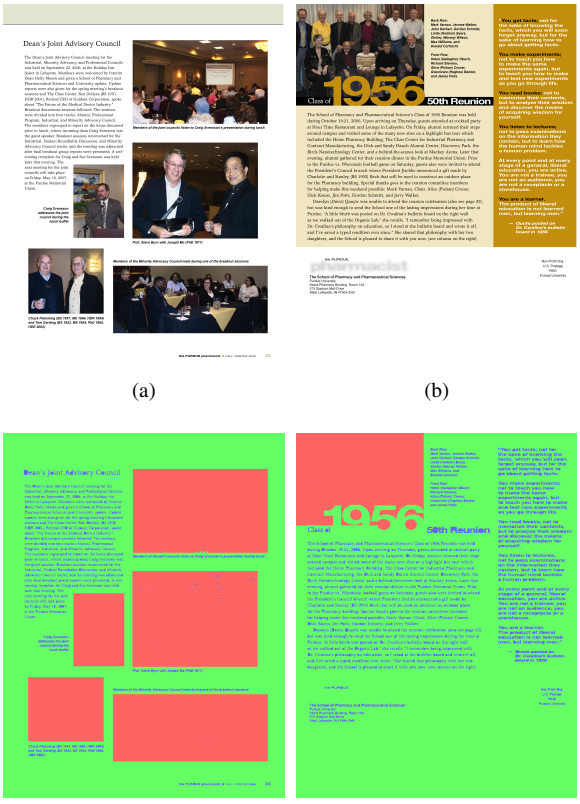
For the classic CCL algorithm, the number of components is bounded by the image size. For our algorithm, it is bounded by the strip size. In addition, because we recycle those out of interest components in each row, the number of components we need to maintain is greatly reduced. In Table 1, we summarize the results from the testing 10 pages. Each page is 6400×4928 pixels. The first test page is Figure 15(a), the second test page is Figure 15(b), and the remaining test pages are not shown here. The comparison indicates that our algorithm can reduce the number of components by an average of 97.46%.

Table 1: Number of components required for the object map generation algorithm based on the classic CCL algorithm and the improved CCL algorithm.

Test Page	Classic	Improved	Reduction
1	330906	8845	97.33%
2	285499	10072	97.47%
3	376013	11137	97.04%
4	206120	6692	96.75%
5	577283	10469	98.19%
6	363944	9675	97.34%
7	773979	12426	98.39%
8	449335	9381	97.19%
9	717915	12722	98.23%
10	217363	7196	96.69 %
Average	429836	9862	97.46 %

4.0 Conclusion

In this paper, we proposed an algorithm that processes an input image one row at a time to generate an object map, which is used for object-oriented halftoning to achieve better print quality. The algorithm is a strip-based process, which only requires a strip of memory buffered between the first-pass and the second-pass, unlike the classic two-pass algorithm that requires the whole page image to be buffered. To further reduce the memory, we recycle unneeded components and labels at each row. The number



(a) (b) (c) (d)

Figure 15: Example object maps generated by our strip-based algorithm. (a)-(b) are the input test pages and (c)-(d) are the object maps generated by our algorithm.

of components is significantly reduced by an average of 97.46% with our algorithm. All the components will be classified as symbol, raster, or vector depending on the number of strips they span and their roughness. To render the image, we use high frequency screening for symbol and raster objects, and low frequency screening for vector objects. What's more, an unbounded symbol object, which behaves like a vector object, will be rendered with low frequency, as well. The proposed algorithm is hardware friendly, and suitable for hardware and ASIC implementation.

References

[1] Chen, Yi-Ting, et al. "Segmentation for better rendering of mixed-content pages." IS&T/SPIE Electronic Imaging. International Society for Optics and Photonics, 2013.

[2] Park, Seong Jun, et al. "Halftone blending between smooth and detail screens to improve print quality with electrophotographic printers." IS&T/SPIE Electronic Imaging. International Society for Optics and Photonics, 2012.

[3] Rosenfeld, Azriel, and John L. Pfaltz. "Sequential operations in digital picture processing." Journal of the ACM (JACM) 13.4 (1966): 471-494.

[4] Kong, Weili, P. Cheng, and Zhuo Bi. "Real-time Sobel edge detector." Proceedings of the 6th PSU-UNS International Conference on Engineering and Technology, 2013.

[5] Lifeng He, Yuyan Chao, and Kenji Suzuki. "A run-based two-scan

labeling algorithm." Image Processing, IEEE Transactions on 17.5 (2008): 749-756.

[6] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001), "Chapter 21: Data structures for Disjoint Sets", Introduction to Algorithms (Second ed.), MIT Press, pp. 498?524, ISBN 0-262-03293-7

[7] Wu, Kesheng, and Ekow Otoo." A simpler proof of the average case complexity of union-find with path compression". Technical Report LBNL-57527, Lawrence Berkeley National Laboratory, 2005.

Authors' Biographies

Zuguang Xiao

Zuguang Xiao is a Ph.D. student at Purdue University in Electrical Engineering. He received his B.S. of Electrical Engineering degree from Purdue University in 2013. His current research mainly focuses on image processing, computer vision, and algorithms.

Mengqi Gao

Mengqi Gao received her B.S. in Electrical Engineering from Purdue University (2014), and is currently studying for the M.S. in Electrical Engineering. Her primary area of research has been image processing, image segmentation, and image quality evaluation.

Lu Wang

Lu Wang received her B.S. degree in Electrical Engineering from Huazhong University of Science and Technology, Wuhan, China, in 2010, and Ph.D. degree from the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, in 2014. After graduating from Purdue, Lu joined Google Inc. as a software engineer, where she is currently working on Android camera technology. Her primary areas of interests include image quality evaluation, computer vision, and image processing.

Jan P. Allebach

Jan P. Allebach is Hewlett-Packard Distinguished Professor of Electrical and Computer Engineering at Purdue University. Allebach is a Fellow of the IEEE, the National Academy of Inventors, the Society for Imaging Science and Technology (IST), and SPIE. He was named Electronic Imaging Scientist of the Year by IS&T and SPIE, and was named Honorary Member of IST, the highest award that IST bestows. He has received the IEEE Daniel E. Noble Award, and is a member of the National Academy of Engineering. He currently serves as an IEEE Signal Processing Society Distinguished Lecturer (2016-2017).