Scan-flood Fill(SCAFF): an Efficient Automatic Precise Region Filling Algorithm for Complicated Regions

Yixuan He[1,2] Tianyi Hu[1,2] Delu Zeng[1*]
{Y.He-48, T.Hu-9}@sms.ed.ac.uk dlzeng@scut.edu.cn
[1] South China University of Technology; [2] The University of Edinburgh.

Abstract

Recently, instant level labeling for supervised machine learning requires a considerable number of filled masks. In this paper, we propose an efficient automatic region filling algorithm for complicated regions. Distinguishing between adjacent connected regions, the Main Filling Process scans through all pixels and fills all the pixels except boundary ones with either exterior or interior label color. In this way, we succeed in classifying all the pixels inside the region except boundary ones in the given image to form two groups: a background group and a mask group. We then set all exterior label pixels to background color, and interior label pixels to mask color. With this algorithm, we are able to generate output masks precisely and efficiently even for complicated regions as long as boundary pixels are given. Experimental results show that the proposed algorithm can generate precise masks that allow for various machine learning tasks such as supervised training. This algorithm can effectively handle multiple regions, complicated 'holes' and regions whose boundaries touch the image border. By testing the algorithm on both toy and practical images, we show that the performance of Scan-flood Fill(SCAFF) has achieved favorable results.

1. Introduction

Nowadays, big datasets are widely used in training models for supervised machine learning in many areas, such as salient object segmentation [1, 2, 3] and lung nodule detection [4]. Although bounding box method can be used to quickly generate training labels [5, 6], instant-level based learning such as segmentation, recognition and detection usually depend on precise masks [7, 1, 2, 3]. To compress the size of the datasets, some of them only annotate the boundary pixels of instances instead of providing all mask pixels. For instance, the LIDC-IDRI dataset [4] widely used for lung nodule detection only provides .XML files that contain the edges of lung nodules. However, in order to achieve better performance on neural networks, especially convolution-based neural networks [8, 9], it is usually important to generate masks, for training. Correspondingly, filling masks from the annotated boundaries is usually a compulsory process in generating ground truth labels for instances. Figure 1 provides an example of how region filling can be used to generate masks for salient object segmentation, given a label image with merely boundary pixels.

Figure 1: A practical example for region filling in generating masks for supervised machine learning.

In general, region filling refers to an approach to fill some bounded regions with given colors. Based on the domain of the graphics they operate in, region filling algorithms can be classified into raster filling and vector filling [10]. In this paper, we focus on raster filling algorithms, *i.e.* on filling regions in raster graphics. This method means filling connected components (defined mostly by 4-connected or 8-connected regions) with required colors. As an important algorithm of Computer Graphics, region filling has various applications in areas such as Computer Aided Design, Realistic graphics, Geographic Information System and Image processing. [11]

*Corresponding author.

Generally, region filling algorithms can be categorized as seed filling and edge filling (or scan-line filling) [12]. However, current algorithms have several shortcomings or potential problems that need to be considered.

Firstly, most of the filling algorithms cannot run automatically. Many seed filling algorithms, such as ordinary boundary fill and flood fill, require at least one known interior pixel within the boundary. Therefore, they are dependent on operator-provided seeds. For complicated objects of interests, multiple seeds may be required and they can be quite hard to be automatically detected[10].

Secondly, some of the filling algorithms cannot deal with complex boundaries. Scan-fill algorithm is capable of filling a polygon's boundaries [13], but may mostly fail to deal with arbitrary boundaries.

Thirdly, it is possible that some of the interior regions' boundary pixels can lie on the border of the image. Hence, it should not be assumed that pixels on the border of an image are all background pixels. The mutual exterior of inner shapes can also be disconnected.

Also, some objects have 'holes' inside. If we simply consider filling connected regions that are not connected to the border of the whole image, we may fail to obtain the precise result.

To overcome these drawbacks, and to reduce the tedious or time-consuming manual work when generating ground truth masks for machine learning, we devise a novel region filling algorithm called **Scan-flood Fill(SCAFF)** algorithm. To show that the algorithm resolves all problems mentioned here, we focus on scenarios described in Table 1.

Table 1: Scenarios for images to be considered in region filling.

Case	#Objects	Touch Border	'Holes' in Object
1	1	No	No
2	1	No	Yes
3	1	Yes	No
4	1	Yes	Yes
5	>1	No	No
6	>1	No	Yes
7	>1	Yes	No
8	>1	Yes	Yes

In summary, the main contributions of our work include: 1) We propose an efficient automatic precise region filling algorithm for complicated regions, robustly dealing with cases listed in Table 1. 2) Scan-flood Fill(SCAFF) algorithm frees operators from having to provide starting seeds for region filling. It can detect seeds automatically. 3) The basic version of our proposed algorithm, EFCI, can deal with regions without 'holes' effectively.

2. Related Works

One of the most common region filling algorithms is seed filling algorithm, or flood filling algorithm, which is usually based on the notion of 4-connectivity or 8-connectivity. Mathematically, let $P = (x, y)$ denote the coordinate of a pixel, then its **4-Connected region** $\mathbf{C_4}(P)$ in the bitmap is defined as

$$\mathbf{C_4}(P) = \{(x, y-1), (x, y+1), (x-1, y), (x+1, y)\}, \quad (1)$$

and its **8-Connected region** $\mathbf{C_8}(\mathbf{P})$ in the bitmap is defined as

$$\mathbf{C_8}(P) = \mathbf{C_4}(P) \cup \{(x-1, y-1), (x+1, y+1), (x-1, y+1), (x+1, y-1)\}, \quad (2)$$

A visual illustration is given in Figure 2.

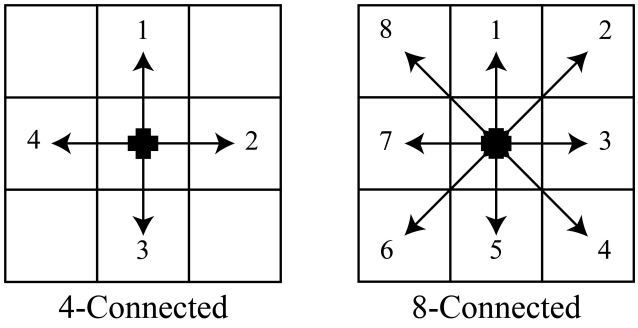


Figure 2: Concept of connectivity. The pixel in the middle of the box is the pixel of interest, and its 4-connected of 8-connected region consists of all pixels that can be reached by arrows starting from it.

Seed filling algorithm starts from a known pixel in a closed area, and recursively finds all the pixels within the connected region of the known pixel. To reach all pixels in each region (normally a 4-connected or 8-connected region as defined above), a stack may be required, and pixels within the region will be visited recursively [14].Depth-first search (DFS) algorithm [15] can be used to fill a certain region from a starting seed. It is efficient in time, but costly in space. Therefore, 'span' filling methods were proposed to reduce the considerable stack required for searching [16]. In 1994, Henrich [17] analyzed several algorithms that saved memory regardless of speed. After that, Yanovsky *et al.* [18] proposed a linear-time constant-space algorithm for the boundary fill problem. However, this algorithm cannot lower the efficiency overhead of revisiting the same nodes. Later, Duo-Le and Ming[11] suggested a 'Marking Method' to tackle the problem of unnecessary pixel revisiting.

Edge filling is another popular approach in region filling, primarily for polygon filling [13, 19]. This algorithm relies

on the detection of the spans of the scan line lying inside the polygon, using the odd-parity rule [20]. However,filling by parity may fail when we obtain the number of line segments incorrectly [21].Although scan line filling algorithms are initially designed for polygons, some authors generalize them to wider applications. For example, Cai [22] took into account the connectivity of pixels in the same region and also made use of a scanning line. Wherever the closed area intersects the scanning line, the scanning line seed filling algorithm takes merely one seed pixel and starts to fill from it to the left and right directions. Nevertheless, repeated judging of the pixel colors and unnecessary backtracking operations lower the efficiency of this algorithm [10]. Vučković *et al*. [23] introduced a generalized iterative scanline fill algorithm useful in real-time applications. In this paper, we adopt some ideas of scan line filling and apply them to label different connected regions.

It is usually difficult to set seeds automatically, and searching for all interior pixels may be time and memory consuming[10, 21]. To automatically find the starting seed, Khayal *et al*. proposed a modified algorithm for seed filling [24]. However, this method needs to discover all contour pixels, and compute their angles. In our algorithm, contour computation is omitted, but we use region labels instead.

Making use of **Connected Component Labeling (CCL)**, which is a fundamental operation in image processing[25], Miao *et al*. [26] proposed a regional filling algorithm based on connected region labeling. However, they did not distinguish between different 'interior' regions, but filled all of them in the same color. Our basic algorithm adopts this idea, and then develops a more precise algorithm using similar ideas to CCL.

3. Proposed Algorithm

In this paper, we propose an automatic(with respect to setting starting seeds for filling) region filling algorithm, built on a basic version. The proposed algorithm is to fill arbitrary regions in a given image, based on the observation that the exterior of a region has to be connected to the border of the whole image. Neither of the basic version and the improved version of our algorithm requires a seed to be given by the operator at the beginning. The basic version of our algorithm can fill arbitrary regions without interior 'holes' precisely, regardless of how many regions there are in the image, and can also handle the case where some regions have some of their boundary pixels on the border of the whole image. The proposed algorithm(*i.e*. the improved version built on the basic version) takes into account potential 'holes' inside a region. Those versions of the algorithm are described as follows.

3.1. Basic Version: Exterior-Fill and Color Inversion (EFCI)

Our method makes use of the property of the exterior mentioned above. The algorithm starts from an image with only boundary pixels in boundary color and the rest in background color. Padding in background color is first added. Then, the mutual exterior of the regions is filled with a temporary exterior label color. In version 1 of the algorithm (where 'floodfill' refers to flood filling algorithm based on the color of the seed, such as OpenCV floodfill [27, 28]), the color of the mutual exterior is set to background color, while the interior is filled with the desired mask color. We call the latter process **Crop-and-'Inverse'** process. 'Inverse' here means to change colors. That is why this basic version of the algorithm can be called **Exterior-Fill and Color Inversion (EFCI).** An example of how the image changes are given in Figure 3.

Algorithm 1: Exterior-Fill and Color Inversion(img)

```
Input: img
Output: resultImg
padImg ← pad img with background color;
padImg ← floodfill(padImg,seed = (0,0)) with exterior
  label color;
croppedImg ← crop padImg to original size, delete
  padding;
resultImg ← croppedImg;
for x in range(resultImg.height) do
    ;                          // color inversion
    for y in range(resultImg.width) do
        if resultImg[x][y] == background color then
            resultImg[x][y] ← mask color;
        if resultImg[x][y] == exterior label color then
            resultImg[x][y] ← background color;
return resultImg
```

In order to start from a pixel in background color, the padding is needed, since the origin of the original image may be a boundary pixel. Besides, multiple regions can be filled within one implementation, without requiring initial seeds inside them to start from. However, since EFCI can only capture the mutual exterior region, which is connected to the origin in the padded image, it cannot handle the case where there are 'holes' inside regions that should not be filled.

3.2. Filling Complicated Regions with Interior 'Holes'

To handle the case with annuli, 'holes' or even more complicated structures in regions, an improved version of

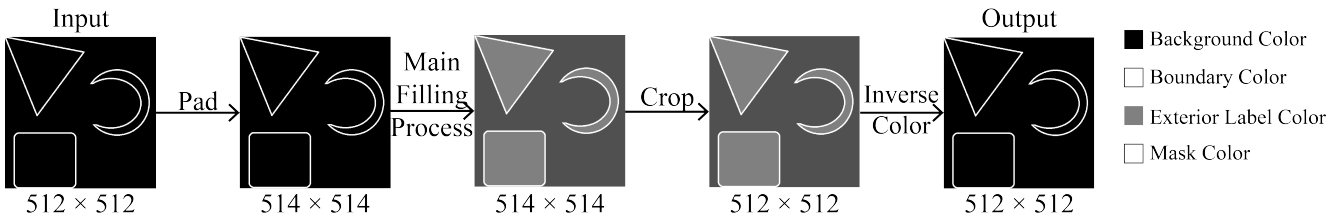


Figure 3: Evolution of an image during the the implementation of EFCI. The flow chart illustrates how EFCI acts on an image: first pads the image, then fills the exterior with exterior label color. After cropping, the algorithm 'inverses' colors to resulting ones.

the algorithm is needed. The intuition is that adjacent connected regions should be treated differently. For example, to fill an image with interior 'holes', as described in Figure 4(a), a desirable filling result should be Figure 4(d), while EFCI tends to give the result in Figure 4(c). If we consider different connected areas to be either background or region to be filled, then adjacent connected regions should be of different types. To be precise, **adjacent connected regions** refer to two connected regions $\mathbf{R_1}, \mathbf{R_2}$ such that:

$$\exists P_1 \in \mathbf{R_1}, P_2 \in \mathbf{R_2} : \overline{P_1P_2} \subseteq \mathbf{R_1} \cup \mathbf{R_2} \cup \mathbf{B}, \quad (3)$$

where $\overline{P_1P_2}$ denotes the line segment between $\mathbf{R_1}$ and $\mathbf{R_2}$, and $\mathbf{B}$ denotes the set of all boundary pixels of the regions in the image.

Intuitively, adjacent connected regions can be merged into one connected region if we take away all boundary pixels. In our example, the connected regions are labelled as 1,2,3,4,5 in Figure 4(b). 1 and 2, 2 and 3 are adjacent connected regions, hence should be treated differently.

To achieve the distinction between different connected regions, we need to introduce the classification of pixels.

3.3. Classification of Pixels

In the proposed algorithm, there are five types of pixels.

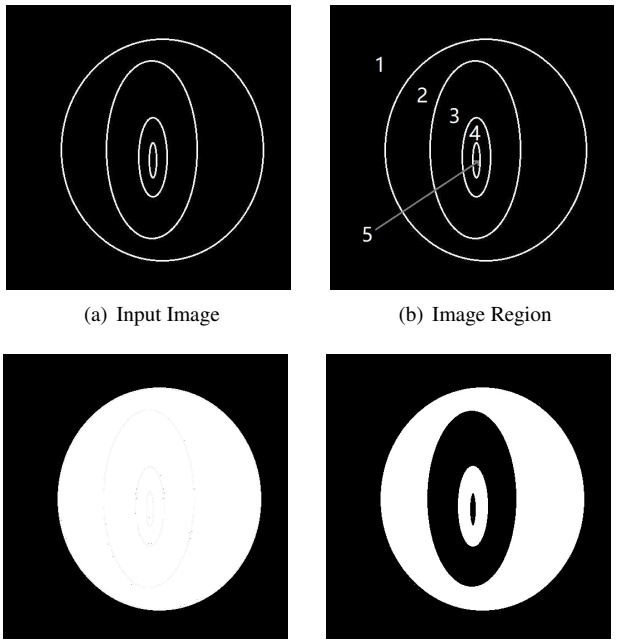
Type 1: Boundary pixels

Boundary pixels are in **boundary color** (pixel value 255 in our experiment), and are used to distinguish between different connected regions, especially during the process of flood filling. Their color is not changed during the implementation of the algorithm.

Type 2: Background pixels

Background pixels are in **background color** (pixel value 0 in our experiment), whose concept varies during the implementation.

In the **Main Filling Process** of our algorithm (line 4 - 15) background pixels correspond to those pixels that have not yet been visited (filled, by either exterior or interior label color). Hence, when visiting a background pixel, the algorithm sets it as the seed of the flood filling algorithm, and then fill its connected region with either exterior or interior label color.



(a) Input Image (b) Image Region

(c) Incorrect Fill (d) Correct Fill

Figure 4: A region filling example with 'holes'. EFCI fails to fill the 'holes' inside boundaries, since it cannot distinguish between adjacent connected regions such as 1 and 2, 2 and 3.

During the **Crop-and-'Inverse' process** (line 16 - 23), exterior label pixels are set to the background color, since their label color merely represents temporary labels, instead of their final color.

Type 3: Exterior Label pixels

Exterior label pixels are in **exterior label color** (pixel value 80 in our experiment). In the Main Filling Process, they are labeled as future 'background pixels', which are to be set back to background color during the Crop-and-'Inverse' process. However, they are colored in a temporary different color from the background, so that the algorithm can tell that they have been visited. A region whose pix-

els are in exterior label color is also useful in determining what color its adjacent connected region should be. In this case, its adjacent connected region should be in interior label color.

Type 4: Interior Label pixels

Interior label pixels are in **interior label color** (pixel value 128 in our experiment). In the Main Filling Process, they are labelled as future 'mask pixels'. However, they are colored in a temporary different color from mask color, which is usually the same as boundary color, so that the algorithm can tell that they are not boundary pixels. A region whose pixels are in interior label color is also useful in determining what color its adjacent connected region should be. In this case, its adjacent connected region should be in exterior label color. Finally, they are set to mask color during the Crop-and-'Inverse' process.

Type 5: Mask pixels

Mask pixels are in **mask color** (pixel value 255 in our experiment), which do not appear until the **Crop-and-'Inverse' process**, when interior label pixels are set to mask color. Those mask pixels mutually make up the resulting masks for the output.

3.4. Proposed Algorithm: Scan-flood Fill(SCAFF)

We illustrate our algorithm below. Given an image with boundary pixels, padding is added. Then we start from the origin of the padded image and use flood filling algorithm to fill the connected region of the origin with label color. After that, we search through all other pixels in the image, setting each unprocessed non-boundary pixel (*i.e*. a background pixel) as a new seed for flood filling, and fill its connected region with a certain color, either exterior or interior label color, based on the color of its adjacent filled connected region. In other to capture the color of its adjacent filled connected region, the algorithm searches backward until reaching a filled pixel, either an exterior or interior label pixel. When all background pixels are filled, we extract the image of the original size from the padded one and transform label color into the background color. Since our algorithm scans through the image to set potential starting seeds for flood filling, and scans backward to determine the color to be filled for the present visited region, we call it **Scan-flood Fill(SCAFF).** An example of how the image changes is given in Figure 5, and the pseudocode is given in algorithm 2.

As illustrated above, this improved version of the region filling algorithm, Scan-flood Fill algorithm, can successfully deal with 'holes', or even more complicated regions.

Algorithm 2: Scan-flood Fill(img)

```
Input: img
Output: resultImg
padImg ← pad img with background color;
seed ← (0,0);
padImg ← floodfill(padImg, seed) with label color;
for x in range(padImg.height) do
    ;            // Main Filling Process
    for y in range(padImg.width) do
        if padImg[x][y] == background color then
            seed ← (x,y), i ← 1;
            while padImg[x][y-i] == boundary color
             do
                i ← i + 1;
            if padImg[x][y-i] == boundary color then
                floodfill(padImg, seed) with filling
                 color;
        if resultImg[x][y] == exterior label color then
            resultImg[x][y] ← background color;
        else
            floodfill(padImg, seed) with label color;
croppedImg ← crop padImg to original size, delete
 padding;
;        // Crop-and-'Inverse' process
resultImg ← croppedImg;
for x in range(resultImg.height) do
    for y in range(resultImg.width) do
        if resultImg[x][y] == exterior label color then
            resultImg[x][y] ← background color;
        if resultImg[x][y] == interior label color then
            resultImg[x][y] ← mask color;
return resultImg
```

4. Experiments

We test both the basic version and our proposed algorithm on a set of images of different sizes, where regions inside have different properties. Comparing their resulting images, we also compute and analyze their running time. All experiments are performed on a PC with Intel(R) Core i7-8550U processor and 16GB RAM.[1]

4.1. Datasets

The input images for our toy experiment are generated by dilation of eight 200×200 basic images. Each of the basic images represents a case in Table 1. Define the property 3-tuple of an image to be $(multiple, border, holes)$, whose entries are all Boolean variables, to represent each of these eight cases. For example, (F, F, T) means that the image has only one object inside, has no regions whose boundaries meet the border of the whole image but has 'holes'

[1]Codes and more related details are given in the following website: https://github.com/SherylHYX/Scan-flood-Fill.

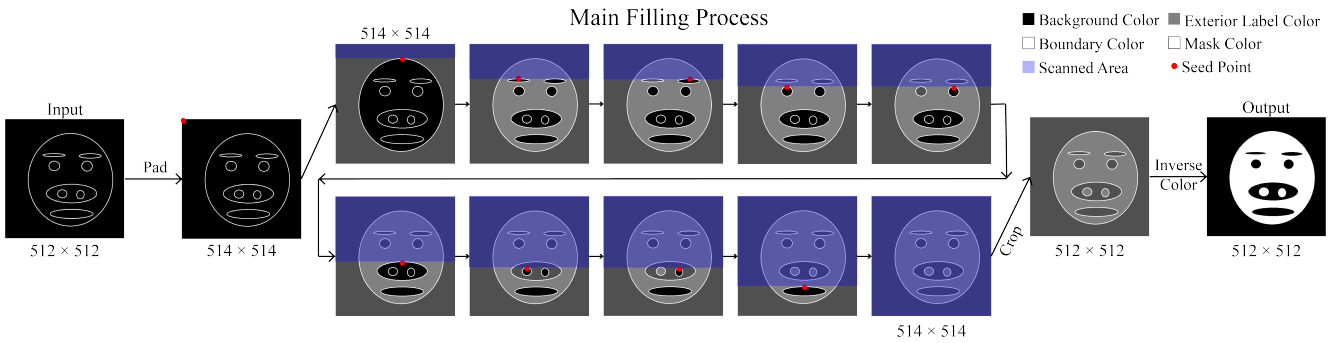


Figure 5: Evolution of an image during the the implementation of Scan-flood Fill(SCAFF) algorithm. The flow chart illustrates how Scan-flood Fill(SCAFF) acts on an image: first pads the image, then labels different connected regions. After cropping, the algorithm sets different regions to resulting colors.The Main Filling Process is illustrated in details, where the algorithm scans through the image, searching for the next seed to start flood fill. The red pixel represents the next seed, and the blue area represents the scanned area.

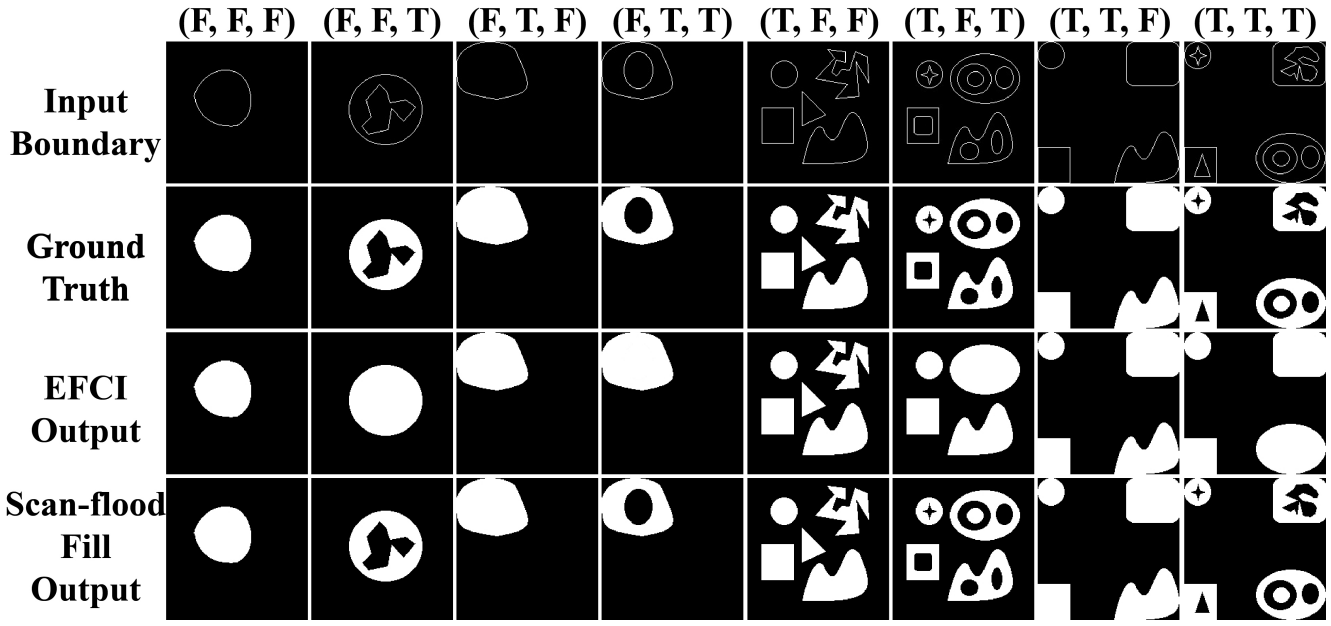


Figure 6: Performance comparison on toy examples

inside regions. This notation will be used to illustrate the comparison on toy examples. To show that our algorithm can also be useful in practical ground truth masks generating process, we also consider images in MSRA10K [29].

4.2. Performance Comparison on Toy Examples

For one thing, most filling algorithms require human-provided starting seeds. For another, among those automatic (with respect to seed selection) region filling algorithms, there are few that share the same filling purpose with us (*i.e.* to fill the regions and to generate masks for machine learning). Hence, we may have different definitions for 'accuracy'. For these reasons, we conduct the performance comparison and evaluation mainly on our proposed algorithm as well as its basic version. We first compare their performance on our toy examples. A visual result comparison is shown in Figure 6.

From the result, we conclude that free from requiring users to provide initial seeds to start from, both algorithms can fill an arbitrary number of irregular regions without 'holes' successfully. Thanks to the padding in the first step of each version of the algorithm, the result will not be influenced by regions whose boundaries lie on the boundary of the whole image. However, when it comes to more complicated regions such as annuli, rings and regions with interior 'holes', only **Scan-flood Fill** can generate desirable results.

Table 2: The time(in seconds) consumed on different image sizes.

Size	Version of Algorithm	
	EFCI (s)	Scan-flood Fill(SCAFF) (s)
200 × 200	0.07766529	0.1186108
400 × 400	0.26739229	0.4606119
600 × 600	0.58787846	1.0168558
800 × 800	1.07662390	1.7586195
1000 × 1000	1.60052500	2.7622317
1200 × 1200	2.32550920	4.1238358
1400 × 1400	3.16818610	5.6745079
1600 × 1600	4.44347420	7.4507671
1800 × 1800	5.45135353	9.4607772
2000 × 2000	6.62144210	11.7155405

4.3. Time complexity evaluation

Table 2 illustrates the average running time of each image in folders with different sizes of input images. For a fair comparison, the time efficiency evaluations of both versions of the algorithm are performed on the same PC. Though Scan-flood Fill takes a bit longer time, the time consumed is not much longer, and the result is more precise.

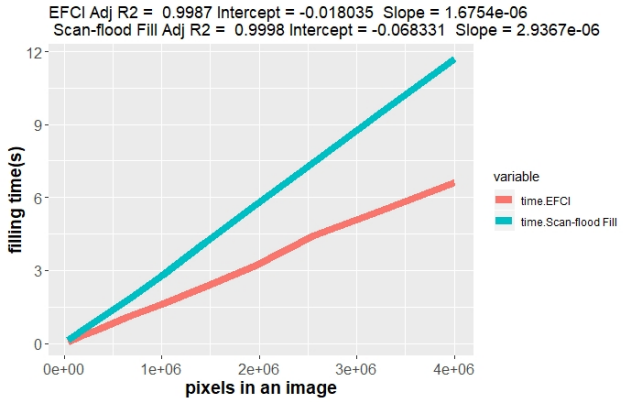


Figure 7: Running time comparison on toy examples.The time consumed is approximately proportional to the number of pixels in the image for each of EFCI and Scan-flood Fill. 'Adj R2' means adjusted R square, 'intercept' means the vertical intercept, and 'slope' means the slope of the regression line.

We also analyze and compare the time complexity of each version of the algorithm. Plotting the number of pixels in an image and the time consumed, we obtain almost straight lines, as shown in Figure 7. By regression, we claim that they both consume almost **linear** time with respect to the number of pixels of an image, and the approximate complexities are both $\mathcal{O}(p)$, where p denotes the number of pixels in the image. Furthermore, in order to save more time for Scan-flood Fill, it is also possible to consider cropping immediately after the first flood fill in the Main Filling Process, *i.e*. exactly after line 3 in Algorithm 2, and then scan through the cropped image, instead of cropping after the whole Main Filling Process.

Table 3: Comparison of quantitative results including F1 score (larger is better) and MAE(Mean Absolute Error, smaller is better), on dataset MSRA10K.

Metric	Version of Algorithm	
	EFCI	Scan-flood Fill
F1 score	0.985600323	0.988181049
MAE	0.005224898	0.004115951

4.4. Practical Use to Generate Ground Truth Masks

We test Scan-flood Fill Algorithm as well as its basic version(EFCI) on 9,918 out of 10,000 images from MSRA10K [29], where edges are relatively easy to be extracted from masks(since this dataset does not provide edge information). Starting from an image with 'only' boundary pixels (*i.e*. masks not yet generated), Scan-flood Fill can generate corresponding masks effectively. To be comparable with ground truth masks given by MSRA10K [29], we set mask color to be boundary color, *i.e*., pixel value 255 in our case. A visual result for some of the images is shown in Figure 8.

We also obtain a comparison of quantitative results including F1 score[30] (larger is better) and MAE[31](Mean Absolute Error, smaller is better), as is given in Table 3. The quantitative results indicate that Scan-flood Fill achieves better performance than EFCI on the given dataset. The difference between them probably lies in the existence of 'holes' within regions to be filled. Besides, since MSRA10K [29] does not provide edge images, and the generating process of edges may result in differences of boundary pixels between ground truth images and generated edge images, it is reasonable that Scan-flood Fill cannot achieve 100 percent accuracy in this case, when compared to GT masks given by the dataset. The Scan-flood Fill results are almost the same as ground truth results, so they can be used as ground truth for supervised learning and would probably not affect training accuracy.

4.5. Advantages of Scan-flood Fill(SCAFF) Algorithm

In previous works, seed filling algorithms such as flood filling algorithm from OpenCV and boundary filling algorithms have been applied to generate filled masks [27, 28]. Although these are used in part of our approach, there exists a considerable difference in that our **starting seeds are automatically provided** by the algorithm instead of being given beforehand. We propose to integrate seed filling algorithms and scan-line filling algorithms, together with the properties of adjacent connected regions, for complicated arbitrary region filling problems.

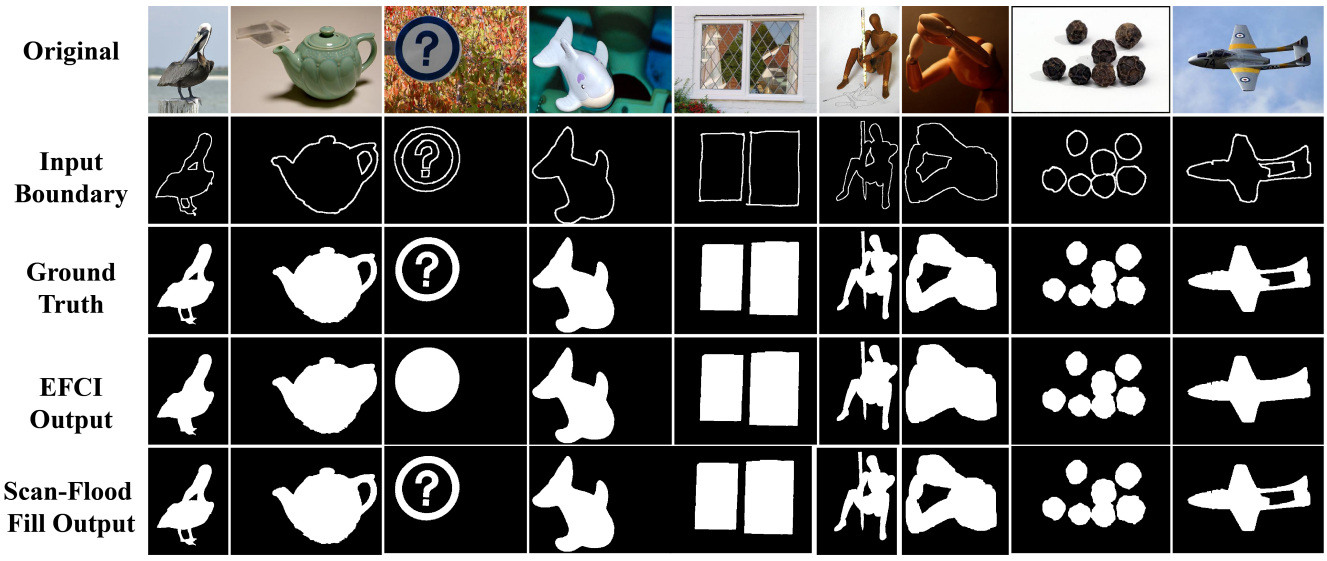


Figure 8: Some practical examples to generate ground truth masks for supervised learning, *i.e.*, from left to right: different images with different types of object shapes; from top to bottom: original input images, boundary images, ground truth images, the corresponding output images with EFCI, and the corresponding output images with Scan-flood Fill(SCAFF).The Scan-flood Fill(SCAFF) results are almost the same as ground truth images, so they can be used as ground truth images and would probably not affect training accuracy.

Besides, Scan-flood Fill(SCAFF) enables us to handle complicated regions such as a pig face and the examples from MSRA10K [29]. This is by virtue of the classification of pixels, especially label pixels, and the Crop-and-'Inverse' process. We compare Scan-flood Fill(SCAFF) with its basic version EFCI for region filling. The results in Figure 6 demonstrates the potential superiority of more precise algorithms taking the relationship between adjacent connected regions into account.

Also, in Scan-flood Fill(SCAFF) algorithm, we do not need to worry about the potential existence of **multiple regions inside an image**. This is because our filling algorithm does not require human-provided starting seeds, but can set starting seeds whenever needed. Moreover, since we start from filling exterior label color to the outermost exterior of an image, which is part of the background for the final result, we are able to avoid being trapped in a small region.

Moreover, padding with background color guarantees the robust filling result regardless of whether some boundaries of the regions in an image lie on the border of the whole image.

5. Conclusion

In this paper, we present an efficient automatic region filling algorithm for arbitrary regions by using a color scheme to assign different labels to adjacent connected regions. The algorithm scans through all pixels in the image, automatically sets the seeds for flood filling, and labels all visited pixels, before the Crop-and-'Inverse' process. The resulting masks distinguish pixels intended to be filled from background efficiently. The Scan-flood Fill(SCAFF) algorithm is effective in generating masks as ground truth images used for supervised model training. Experiments on various types of images clearly demonstrate the effectiveness of our algorithm, and robustness when handling multiple regions, complicated 'holes' and regions intersecting image border. Besides, as the problem confronted is to judge the interior pixels bounded by some arbitrary shapes, we still consider the proposed algorithm to be eligible in giving some possible potential insight into overcoming the related graphics or computational geometry problems.

6. Acknowledgement

This work was supported in part by grants from National Science Foundation of China (No.61571005, No.61811530271), the China Scholarship Council (CSC NO.201806155037), the Science and Technology Research Program of Guangzhou, China (No.201804010429), the Fundamental Research Funds for the Central Universities, SCUT (No.2018MS57).

References

[1] Z. Ren, S. Gao, L.-T. Chia, and I. W.-H. Tsang, "Region-based saliency detection and its application in object recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 5, pp. 769–779, 2014.

[2] L. Wang, H. Lu, X. Ruan, and M.-H. Yang, "Deep networks for saliency detection via local estimation and global search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3183–3192, 2015.

[3] F. Perazzi, P. Krähenbühl, Y. Pritch, and A. Hornung, "Saliency filters: Contrast based filtering for salient region detection," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 733–740, IEEE, 2012.

[4] M. F. Mcnitt-Gray, S. G. Armato, C. R. Meyer, A. P. Reeves, G. Mclennan, R. C. Pais, J. Freymann, M. S. Brown, R. M. Engelmann, P. H. Bland, G. E. Laderach, C. Piker, J. Guo, Z. Towfic, D. P.-Y. Qing, D. F. Yankelevitz, D. R. Aberle, E. J. van Beek, H. Macmahon, E. A. Kazerooni, B. Y. Croft, and L. P. Clarke, "The lung image database consortium (lidc) data collection process for nodule detection and annotation," *Academic Radiology*, vol. 14, no. 12, pp. 1464–1474, 2007.

[5] B. Zhong, S. Pan, H. Zhang, T. Wang, J. Du, D. Chen, and L. Cao, "Convolutional deep belief networks for single-cell/object tracking in computational biology and computer vision," *BioMed Research International*, vol. 2016, 2016.

[6] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask r-cnn," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[7] S. Cai, J. Huang, D. Zeng, X. Ding, and J. Paisley, "Menet: a metric expression network for salient object segmentation," *arXiv preprint arXiv:1805.05638*, 2018.

[8] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241, Springer, 2015.

[9] N. Liu and J. Han, "Dhsnet: Deep hierarchical saliency network for salient object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 678–686, 2016.

[10] M. C. Codrea and O. S. Nevalainen, "Note: An algorithm for contour-based region filling," *Computers & Graphics*, vol. 29, no. 3, pp. 441–450, 2005.

[11] F. Duo-Le and Z. Ming, "A new fast region filling algorithm based on cross searching method," vol. 202, pp. 380–387, 2011.

[12] T. Pavlidis, *Algorithms for graphics and image processing*. Digital system design series, Rockville, Md.: Computer Science Press, 1981.

[13] D. M. R, "Scan line generator for area fill of extensible polygons," 1997.

[14] X. Li and L. Huang, "New region filling algorithm based on chain codes description," in *2010 3rd International Congress on Image and Signal Processing*, vol. 6, pp. 2806–2809, IEEE, 2010.

[15] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972.

[16] J. D. Foley, F. D. Van, A. Van Dam, S. K. Feiner, J. F. Hughes, J. HUGHES, and E. ANGEL, *Computer graphics: principles and practice*, vol. 12110. Addison-Wesley Professional, 1996.

[17] D. Henrich, "Space-efficient region filling in raster graphics," *The Visual computer*, vol. 10, no. 4, pp. 205–215, 1994.

[18] V. M. Yanovsky, I. A. Wagner, and A. M. Bruckstein, "A linear-time constant-space algorithm for the boundary fill problem," *The Computer Journal*, vol. 50, no. 4, pp. 473–477, 2007.

[19] S. Anitha and D. Evangeline, "An efficient fence fill algorithm using inside-outside test," *International Journal of Advanced Research in Computer science and Software Engineering*, vol. 3, no. 11, 2013.

[20] R. C. Gonzalez, *Digital image processing*. Harlow, United Kingdom: Pearson Education Limited, fourth edition, global edition.. ed., 2018.

[21] M. Ren, W. Yang, and J. Yang, "A new and fast contour-filling algorithm," *Pattern Recognition*, vol. 38, no. 12, pp. 2564–2577, 2005.

[22] Z. Cai, "Restoration of binary images using contour direction chain codes description," *Computer Vision, Graphics, and Image Processing*, vol. 41, no. 1, pp. 101–106, 1988.

[23] V. Vučković, B. Arizanović, and S. Le Blond, "Generalized n-way iterative scanline fill algorithm for real-time applications," *Journal of Real-Time Image Processing*, pp. 1–19, 2017.

[24] M. Khayal, A. Khan, S. Bashir, F. Khan, and S. Aslam, "Modified new algorithm for seed filling," *Journal of Theoretical and Applied Information Technology*, vol. 26, no. 1, pp. 28–32, 2011.

[25] F. Spagnolo, F. Frustaci, S. Perri, and P. Corsonello, "An efficient connected component labeling architecture for embedded systems," *Journal of Low Power Electronics and Applications*, vol. 8, no. 1, 2018.

[26] M. Longyuan, Y. Zhenglin, and W. Zhen, "Regional filling algorithm based on connected region labeling," *Journal of Changchun University of Science and Technology*, vol. 41, no. 4, pp. 114–117, 2018.

[27] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[28] J. Howse, *OpenCV computer vision with python*. Packt Publishing Ltd, 2013.

[29] M.-M. Cheng, N. J. Mitra, X. Huang, P. H. Torr, and S.-M. Hu, "Global contrast based salient region detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 569–582, 2015.

[30] C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation," in *European Conference on Information Retrieval*, pp. 345–359, Springer, 2005.

[31] E. J. Coyle and J.-H. Lin, "Stack filters and the mean absolute error criterion," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 8, pp. 1244–1254, 1988.