

## Fast Texture Segmentation using Genetic Programming

**Andy Song**

School of Computer Science and Information Technology  
RMIT University, GPO Box 2476V  
Melbourne, Australia  
asong@cs.rmit.edu.au

**Vic Ciesielski**

School of Computer Science and Information Technology  
RMIT University, GPO Box 2476V  
Melbourne, Australia  
vc@cs.rmit.edu.au

**Abstract-** This paper presents a method which extends the use of genetic programming (GP) to a complex domain, texture segmentation. By this method, segmentation tasks are performed by texture classifiers which are evolved by GP. Small cutouts sampled from images of various textures are used for the evolution. The generated classifiers directly use pixel values as input. Based on these classifiers an algorithm which uses a voting strategy to partition texture regions is developed.

The results of the investigation indicate that the proposed method is able to accurately identify the boundaries between different texture regions, even if the boundaries are not regular. The method can segment two textures as well as multiple textures. Furthermore fast segmentation can be achieved. The speed of the proposed texture segmentation method can be a hundred times faster than conventional methods.

### 1 Introduction

Texture segmentation and texture classification are two important aspects of texture analysis [15]. Texture segmentation, which can be considered as an extension of texture classification, is the overlapping area between texture analysis and image segmentation. The aim of texture segmentation is to partition an image into regions based on differences of textural appearance (see examples in Figures 2 and 3). Texture analysis and segmentation in general are both important in machine vision applications such as scene analysis and target recognition. However both of them are difficult tasks in this area and remain unsolved. In this paper, we approach them by using Genetic Programming (GP).

GP has been demonstrated to be a flexible method of problem solving for a diverse range of complex problems [7]. One task that GP has been applied to is classification. During the evolutionary process a good classifier, which is actually a program expected to be able to discriminate different classes, is searched for. In previous investigations genetic programming techniques have been shown to be capable of producing accurate classifiers in a variety of domains such as medical diagnosis [8], object detection [14, 17] and image analysis [10, 11].

In our previous studies, GP was used as a new method for texture classification directly based on pixel inputs [12, 13]. The results from these studies indicate that the texture classifiers evolved by GP can accurately classify textures. It has also been shown that the texture classifiers generated by GP are small. Furthermore, these texture classifiers are evolved without domain knowledge. In other words, feature extraction, which is computationally expensive, can be avoided. This indicates that fast execution can be achieved by these classifiers.

In this study, we extend the use of evolved classifiers to develop a new texture segmentation method. The goals of this study are

- To investigate the applicability of such classifiers in texture segmentation.
- To develop a framework based on these classifiers to accurately segment images containing two or more textures.
- To develop a fast texture segmentation method, because segmentation speed is essential for real world applications especially real-time applications and applications under resource-constraint environment.

### 2 Generating Classifiers

The method of generating texture classifiers is called the single-step approach in our work because it does not require a feature extraction phase for the classification. In contrast conventional texture classification methods need two major steps, feature extraction plus classification.

Generally, classifiers can be represented as program trees. One method of performing classification using GP is dynamic range selection, where a classifier returns numeric values and the class is determined by the output of the classifier [8]. Furthermore, the ranges for class labels over a set of classes are determined dynamically [8]. Our study focuses on binary classification, so the output of a classifier is interpreted as either "Class 1" or "Class 2". The dynamic range selection method of classifier representation was shown to be capable of producing accurate results over

Function	Description
Plus	Arithmetic addition
Minus	Arithmetic subtraction
Mult	Arithmetic multiplication
Div	Protected arithmetic division (divide by zero returns zero)
IF	Conditional. If arg1 is true return arg2, otherwise return arg3
<=	True if arg1 is <= arg2
>=	True if arg1 is >= arg2
=	True if arg1 is equal to arg2
Between	True if the value of arg1 is between arg2 and arg3

Table 1: Function Set

Name	Description
Random(-1, 1)	Random number in $[-1, 1]$
$V[x, y]$	Pixel Value of $(x, y)$

Table 2: Terminal Set

a variety of datasets in the medical and image processing domains [8, 13].

Tables 1 and 2 list the function and terminal sets. The function set consists only of simple arithmetic and logic operators. No computationally expensive functions such as image processing functions are included. The terminal set consists of only random numbers and pixel values. So the classifiers use pixel values directly rather than feature values as input.

The fitness measure for texture classification is straightforward. The performance can be determined by the success rate of the program, which is, in this case, the classification accuracy. As a result the fitness value can be expressed by the following formula:

$$f = \frac{\text{Number of Successes}}{\text{TOTAL}} = \frac{\text{TP} + \text{TN}}{\text{TOTAL}} \times 100\% \quad (1)$$

where TP is the number of true positives, TN is the number of true negatives and TOTAL is the total number of cases in the training data set.

A training data set is a collection of small cutouts sampled from the images of both classes. These cutouts are also called sub-images. In case of classifying one texture against another texture, about one thousand sub-images are sampled from an image of one texture to form ‘‘Class 1’’, and an equal number of sub-images are sampled from an image of another texture to form ‘‘Class 2’’. In the case of classifying one texture against a group of other textures, the one thousand sub-images of ‘‘Class 2’’ are a mixture of small cutouts sampled from the textures belonging to the group. For example, to train a classifier for differentiating texture

A against texture B, C and D, one thousand sub-images will be sampled from the image of texture A, and about 333 sub-images will be sampled from the images of textures B, C and D respectively to form ‘‘Class 2’’.

Our runs used a population size of 500 individuals. The termination criteria for each run was either perfect classification—where classification accuracy was 100%—or after 50 generations had been processed. The crossover operation accounted for 90% of the breeding pool and elitist reproduction was used for the remaining 10%. The mutation operator was not used. The programs were generated with an initial maximum depth of 6, with overall program depth limited to 17. The classifiers with highest training accuracy during the evolution were used for the later segmentation experiments.

Our experiments used two kinds of textures, bitmap patterns and Brodatz texture. Bitmap patterns represent simple textures. These texture images are composed of pixels of only two values, **0** or **1**. Brodatz textures represent difficult textures. Each pixel is an 8-bit grey level intensity value. They originate from a photographic album published in 1966 [1]. This album is now a kind of *de facto* standard database in texture-related research [9].

### 3 Texture Segmentation

Texture segmentation techniques can be categorized in several ways such as whether they use supervised learning or unsupervised learning; whether they focus more on identifying the boundaries or more on identifying regions and whether they use a split-and-merge method or a voting strategy.

Our proposed segmentation algorithm is shown in Figure 1. The first step is evolving texture classifiers by the single-step approach described in previous section. So our approach is a supervised method because sub-images need to be labeled for training. Although GP can be used for unsupervised learning such as [4], it is more practical to use it as a means of supervised learning in our study. This algorithm can be considered as a region-based approach, because it is based on labeling regions rather than differentiating two adjacent regions. A voting strategy is used to determine the class of a pixel.

In the following sections, some examples of texture segmentation tasks are presented, including segmenting multiple regions and segmenting regions with complex boundaries. Both bitmap patterns and Brodatz textures are used in the experiments. During the training, ten-fold cross validation was performed. The training and test accuracies of these classifiers measured in cross validation are all recorded. All the segmentation experiments were conducted on a SUN Sparc workstation. The CPU runtimes on this machine are presented with the corresponding output images.

---

Input:  $T1, T2$  textures for which example images are available  
 $I$  an image containing a number of regions of textures  $T1$  and  $T2$  and no other textures  
 $w, h$  width and height of  $I$   
 $n$  sub-image size  
 $d$  step size for moving window,  $1 < d \leq n/2$

Output:  $O$  a two colour image  $O_{ij} = colour1$  for texture 1 and  $O_{ij} = colour2$  for texture 2

---

1. Generate a single-step classifier which uses a sub-image of size  $n \times n$ .
  2. Use the generated classifier to sweep the input image  $I$ .
    - (a) Start at the top-left corner of the image.
    - (b) Sample a sub-image by a  $n \times n$  window and classify it by the generated classifier.
    - (c) Label all the pixels in this window with the class label output of the classifier.
    - (d) Move the window  $d$  pixels right and repeat 2(b) and 2(c), until the window reaches the right boundary of the image.
    - (e) Reposition the window to the left edge of the image and move down by  $d$  pixels, then repeat from step 2(b) until the whole image has been completely sampled.
  3. Generate the output image which only contains the regions.
    - (a) Label each pixel based on voting. For example, if the majority of classifier outputs for a pixel is texture  $T1$ , then this pixel is labeled as  $T1$ .
    - (b) Assign each pixel a color based on its class label.
    - (c) Output the generated image  $O$ .
- 

Figure 1: Segmentation Algorithm for Two Textures

### 3.1 Two textures with Multiple Regions

Figure 2 shows an input image containing two bitmap patterns (vertical lines and horizontal lines) and the segmented output image. It also shows the parameter values used and the results. The texture regions in the output image are iden-

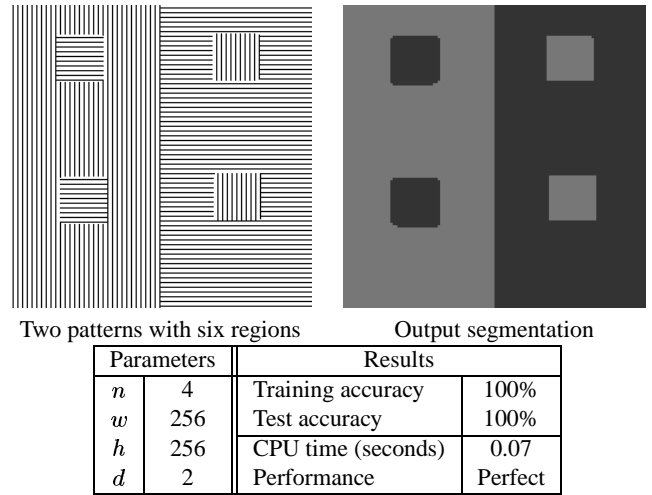


Figure 2: Segmenting Two Patterns with Multiple Regions

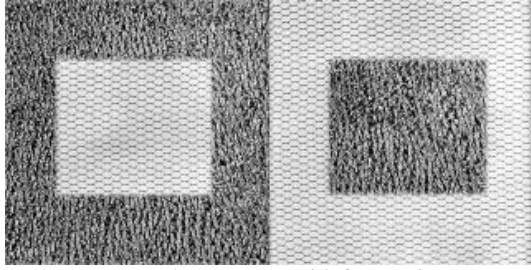
tified by two different intensity levels, gray and black. The segmentation performance was evaluated based on visual comparison between the original image and the corresponding output. Our study aims to investigate the methodology rather than to compare it with other methods in terms of segmentation accuracy. Therefore a quantitative measurement of segmentation performance is not used.

The results indicate that the classifier generated by the single-step approach is very fast on segmenting a  $256 \times 256$  binary image. The boundary between the left half and the right half in the output image is precisely the boundary in the original image. The four small patches of “alien” textures are also identified and accurately located although their boundaries are not perfectly square.

In the case of Brodatz textures shown in Figure 3, the separation of the different textural regions<sup>1</sup> is still very accurate although the boundaries are not perfectly delineated as in the original image. The segmentation required 0.37 seconds of CPU time. This is much longer than for segmenting the two Bitmap patterns (shown in Figure 2) even though the Brodatz image has fewer pixels than the binary image. This is mainly because the pixels are 8-bit in Brodatz textures rather than 1-bit in bitmap patterns, the window size  $d$  in this case is 16 rather than 4, and the classifier generated for Brodatz textures is larger than that for Bitmap patterns.

The resulting segments indicate that the proposed method is able to separate regions of different textures and it is also able to recognize textures. In the input image of Figure 2, the texture of the two small patches on the left is the same as the “background” of the right half of the image. Correspondingly, the two small regions in the left

<sup>1</sup>The two textures are known as D24 and D34 in the Brodatz Album.



Two Brodatz textures with four regions



Output segmentation

Paramters		Results	
$n$	16	Training accuracy	94.05%
$w$	320	Test accuracy	93.00%
$h$	160	CPU time (seconds)	0.37
$d$	2	Performance	Almost Perfect

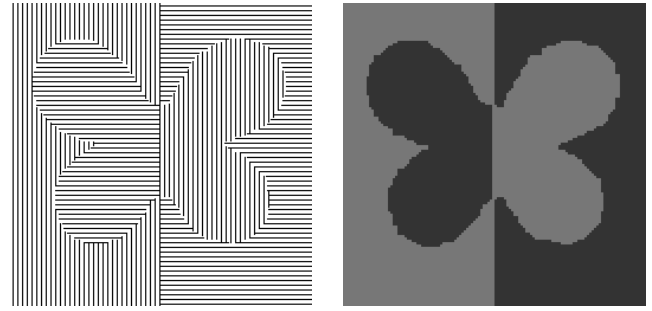
Figure 3: Segmenting Two Brodatz Textures with Multiple Regions

half of the output image are marked with the same color as the right sided “background”, establishing that these regions were recognized as having the same texture. Similarly for the two patches on the right. The output image shown in Figure 3, in which different areas of the same texture are marked with the same color, provides further evidence for the ability of the proposed method to separate and recognize textures.

### 3.2 Regions with Complex Boundaries

The region boundaries in the previous segmentation tasks are always straight, either horizontal or vertical. Such boundaries might be easier to handle because the strictly square sampling window is used. However straight boundaries are rarely found in real life applications. Therefore, more variations of input images are introduced to examine the capacity of our proposed method to identify irregular boundaries.

The input image in Figure 4 has a butterfly-shaped boundary. The left wing of the butterfly contains the same texture as the background of the right half of the image, while its right wing has the same texture as the left sided background. The same classifier used for segmenting im-



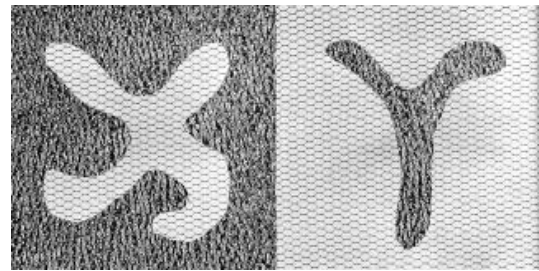
Two patterns

Output segmentation

CPU time (seconds)	0.07
Segmentation Performance	Very Good

Figure 4: Segmenting Two Patterns with Complex Region Boundaries

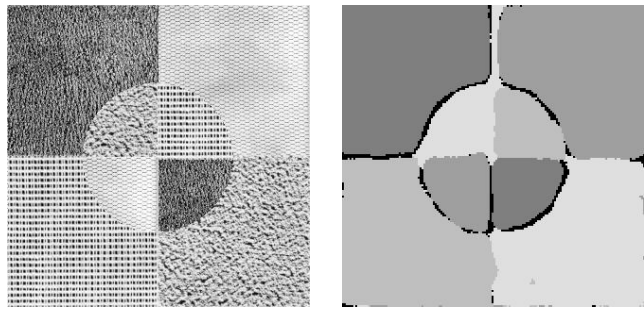
age in Figure 2 was also used here. As expected, the CPU runtime is independent of the content of target images. The output image clearly identifies the four regions in the image. The segmented boundaries of the two wings are not as smooth as in the original image. Adjacent areas of two texture regions are usually problematic areas especially when the boundary is not straight. However the output boundaries closely resemble the actual ones, so the segmentation outcome is considered as very good.



Free Hand X and Y

Figure 5: Segmenting Two Brodatz Textures with Irregular Boundaries

In Figure 5 there are two free hand letters. The segmented regions show that the textural regions have been rec-



Original Image, size  $320 \times 320$

Segmentation Output

Parameters		Results	
$n$	16	Each Color indicates one texture	
$w$	320	Black: the unidentified areas	
$h$	320	CPU time (sec)	2.58
$d$	2	Performance	Good

Figure 7: Segmenting Four Textures, D21, D24, D34, D57

ognized and the generated boundaries are very close to the actual boundaries in the input images. The segmentation outcomes are considered as very good.

## 4 Segmenting Multiple Textures

The algorithm shown in Section 2 is specific to two-texture problems. It can be extended for multiple textures segmentation by decomposing such a task into multiple binary-class problems. Most of the published work on using GP for image related tasks focus on binary-class problems.

### 4.1 Segmentation Algorithm for Multiple Textures

Figure 6 shows the algorithm for segmenting images with multiple textures. This algorithm decomposes the problem of classifying  $T_1, T_2, T_3 \dots T_n$  to classifying  $T_1$  with  $\{T_2, T_3 \dots T_n\}$ , classifying  $T_2$  with  $\{T_1, T_3 \dots T_n\}$  ... until classifying  $T_n$  with  $\{T_1, T_2 \dots T_{n-1}\}$ .

There are several ways to combine multiple classifiers such as bagging, boosting and stacking [16]. The multi-classifier combination used in this algorithm is similar to boosting which considers the accuracy of each classifier. However, it ranks classifiers based on their test accuracy. The final decision is not made by the aggregation of classifiers with weights but by the one with the highest rank. The most suitable method of combining multiple classifiers for segmentation tasks will not be fully investigated here.

### 4.2 Segmenting Four Textures

An image composed of four Brodatz textures<sup>2</sup> is illustrated in Figure 7.

<sup>2</sup>They are known as D24, D34, D21 and D57 respectively.

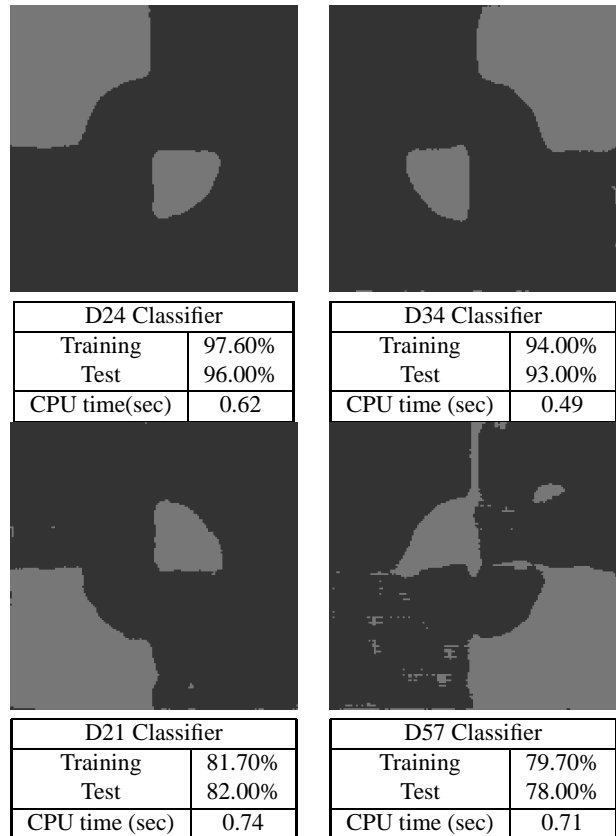


Figure 8: Outputs of Four Classifiers

Segmentation was accomplished by the combination of four classifiers. The output image presented clearly shows four distinct segments of different colors separated by boundaries that closely resemble the actual boundaries. The black areas represent pixels that have been rejected and marked as “N” by all the classifiers. Most of the black pixels are located close to boundaries which are known difficult areas. The input image size is  $320 \times 320$  which is twice the size of images previously used for two Brodatz textures.

To assess the performance of the classifiers, each was applied to the segmentation task individually. The experimental results are presented in Figure 8, along with the training accuracy and test accuracy of each classifier. The regions claimed by a classifier as “its own” texture are painted gray and regions not claimed by it are painted black. Other than D57, the classifiers are fairly accurate in identifying their own regions. In the case of D57, the classifier recognized the entire region containing D57 despite a poor training accuracy of only 79.70%. This classifier also marked some areas which are not D57, that is, it made some false positive errors. Most of the false positives of D57 are eliminated in the final output because the classifiers for D21 and D34 which also claimed these areas were ranked higher in terms

---

Input:	$m$	number of textures
	$T = \{T_1, T_2 \dots T_m\}$	textures for which example images are available
	$I$	an image containing a number of regions of textures $T', T' \subset T$
	$w, h$	width and height of $I$
	$n$	sub-image size
	$d$	step size for moving window, $1 < d \leq n/2$
Output:	$O$	an image of $m$ colours where each pixel is a colour corresponding to the texture at that pixel in the original image

1. Generate  $m$  single-step classifiers which use a sub-image of size  $n \times n$ . Each classifier is to distinguish one texture  $T_i$  (class 1) from other textures  $\{T_j | 1 \leq j \leq m, j \neq i\}$  (class 2). These classifiers are labeled as  $Classifier_1, Classifier_2, \dots, Classifier_m$
  2. Repeat the following process for each classifier from  $Classifier_1$  to  $Classifier_m$ :
    - (a) Use  $Classifier_i$  to sweep the input image  $I$ .
      - i. Start at the top-left corner of the image.
      - ii. Sample a sub-image by a  $n \times n$  window and classify it by  $Classifier_i$ .
      - iii. Label all the pixels in this window with the class label output by the classifier.
      - iv. Move the window  $d$  pixels right and repeat 2(a)[ii] and 2(a)[iii], until the window reaches the right boundary of the image.
      - v. Reposition the window to the left edge of the image and move down by  $d$  pixels, then repeat from 2(a)[ii] until the whole image has been completely sampled.
    - (b) Generate the output image which only contains the regions.
      - i. Label each pixel based on voting. If the majority of the classifier outputs for a pixel is texture  $T_i$  (class 1), then this pixel is labeled as  $T_i$ . Otherwise the pixel is labeled as  $N$  (class 2).
      - ii. Record the label decided in step 2(b)[i] for each pixel.
  3. Assemble the votes given by each classifier to each pixel.
    - If one pixel is classified as  $T_i$  by  $Classifier_i$  and  $N$  by all other classifiers, then the final class of the pixel will be  $T_i$ .
    - If one pixel is classified as  $N$  by all the classifiers, then the final class of the pixel will be "N".
    - If one pixel is claimed by multiple classifiers, eg. it is classified as  $T_i$  by  $Classifier_i$  and classified as  $T_j$  by  $Classifier_j$ , then the final decision will be made by the classifier which has the highest test accuracy. If  $Classifier_i$  is more accurate than  $Classifier_j$  on test data, then the pixel will be considered as  $T_i$  rather than  $T_j$ .
  4. Produce the output image  $O$  based on the final decisions. Each class label corresponds to a separate color while the pixels marked as  $N$  will be black.
- 

Figure 6: Segmentation Algorithm for Multiple Textures

of their accuracy.

## 5 Segmentation Speed

The classifiers generated by the single-step approach are small. No classifier for Brodatz textures was observed to have more than 700 functions, which are simple operators like  $+$ ,  $-$ ,  $*$ ,  $/$  and  $IF$ . This characteristic enables a fast voting strategy which can lead to much faster segmentation than the conventional “feature extraction plus classification” approaches. In comparison, feature extraction algorithms are much more complex. Using Haralick features [5] as an example, at least one  $256 \times 256$  co-occurrence matrix needs to be generated in computing the features for an image with 256 gray levels. One of the simplest Haralick features, Energy, is computed as

$$\sum_{i=0}^{255} \sum_{j=0}^{255} M^2(i, j)$$

where  $M(i, j)$  denotes one cell of the matrix. Such a calculation requires  $256 \times 256$  repetitions of “+” and “\*”. Therefore more than 100,000 operations are needed.

Consider a hypothetical situation where a segmentation method has been developed based on the conventional approach. If the method uses a voting strategy, the window is  $16 \times 16$ , the step size  $d$  is 2, then the processing time for a  $256 \times 256$  gray level image is approximately<sup>3</sup>

$$\left( \frac{256 - 16}{2} + 1 \right)^2 \times 0.7 \approx 10000(\text{seconds})$$

If a split-and-merge approach is used and the split is limited to only three levels, then the total CPU time for computing these features can be estimated as<sup>4</sup>

$$4 \times 4.58 + 16 \times 4.10 + 64 \times 1.70 = 192.72(\text{seconds})$$

Clearly, whether the voting strategy or the split-and-merge strategy is used, the times taken for computing these features are much longer than 2.58 seconds, the time for segmenting a  $320 \times 320$  image by our proposed algorithm. Jain and Kalle reported 122 seconds of processing time on a SUN Sparc-10 workstation to segment a  $256 \times 256$  gray-level image using Gabor filters and 109 seconds processing time by their proposed method [6]. Chang *et al.* evaluated different texture segmentation algorithms and reported the runtimes of “feature extraction plus classification” on a

<sup>3</sup>0.7 seconds is the time needed to compute Haralick features on the same SUN Sparc workstation for a  $16 \times 16$  gray-level image.

<sup>4</sup>The times for computing Haralick features for images of size  $128 \times 128$ ,  $64 \times 64$  and  $32 \times 32$  on the same machine are 4.58, 4.10 and 1.70 seconds respectively.

SUN Ultra Sparc. All of the times were more than 28 minutes [2]. In a recent publication, Chen and Chen proposed a new feature for fast texture segmentation [3]. The runtime for their experiment of segmenting  $256 \times 256$  images on a Pentium III-500 PC was about 1 minute. Although the runtimes reported in the literature are measured under different conditions and not suitable for direct comparison, it is clear that the proposed method requires much less computation time since there is no computational expensive feature extraction.

## 6 Conclusion

The segmentation results presented show that a fast and accurate texture segmentation method can be developed based on classifiers evolved by GP. This method can produce excellent results in various kinds of segmentation tasks, both in bitmap patterns and Brodatz textures, and with two classes of textures or with multiple classes of textures. By using voting strategies, this method can produce relatively smooth boundaries and handle complex boundaries.

The advantages and disadvantages of this new segmentation method are briefly summarized below. A more thorough investigation of this method is left for further work.

Advantages:

- The segmentation method is very fast. This makes it well suited for real time applications.
- The method can handle complex shapes and produce relatively smooth boundaries.
- The constituent classifiers do not have to be perfectly accurate.
- Different classifiers can be evolved in different runs and contribute to an ensemble. This can enhance segmentation performance.
- The basic algorithm can readily be extended to segmentation of images with three or more textures.

Drawbacks:

- The current approach is a kind of supervised learning, therefore it is not directly suitable for unsupervised segmentation. Prior to segmentation, it is essential to know how many classes of textures are in the images and to have examples of each.
- Although the generated classifiers are fast, the time taken for the evolutionary process to create these classifiers can be quite long, from a few hours to a few days.

- Although the method is general, the generated classifiers are problem-specific. For different applications different classifiers need to be trained.

This study introduced the use of genetic programming for texture segmentation. It opens up a new application area for GP, in which many interesting topics can be explored in the future. These include such as how to achieve an ensemble of classifiers more efficiently, whether it is possible to address unsupervised texture segmentation by GP and how to construct a generalized methodology for using GP in the texture segmentation area.

## Bibliography

- [1] Phil Brodatz. *Textures: A Photographic Album for Artists and Designers*. Dover, NY, 1966.
- [2] K.I. Chang, K.W. Bowyer and M. Sivagurunath. Evaluation of texture segmentation algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1999*, Volume 1, pages 294–299, 1999.
- [3] Kan-Min Chen and Shu-Yuan Chen. Color texture segmentation using feature distributions. *Pattern Recognition Letters*, Volume 23, Number 7, pages 755–771, May 2002.
- [4] I. De Falco, E. Tarantino and A. Della Cioppa. Unsupervised spectral pattern recognition for multispectral images by means of a genetic programming approach. In *Proceedings of the 2002 Congress on Evolutionary Computation, 2002*, pages 231–236, May 2002.
- [5] R. M. Haralick, K. Shanmugam and I. Dinstein. Textural features for image classification. *IEEE Transactions On Systems, Man, and Cybernetics*, Volume SMC-3, Number 6, pages 610–621, November 1973.
- [6] Anil K. Jain and Kalle Karu. Learning texture discrimination masks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 18, pages 195–205, 1996.
- [7] John R. Koza. *Genetic Programming III: Aarwinian Invention and Problem Solving*. Morgan Kaufmann, San Francisco, 1999.
- [8] Thomas Loveard and Vic Ciesielski. Representing classification problems in genetic programming. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 27–30. IEEE Neural Network Council (NNC), Evolutionary Programming Society (EPS), Institution of Electrical Engineers (IEE), IEEE Press, May 27-30 2001.
- [9] R.W. Picard, T. Kabir and F Liu. Real-time recognition with the entire Brodatz texture database. In *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR '93., 1993 IEEE Computer Society Conference on*, pages 638–639, 1993.
- [10] Riccardo Poli. Genetic programming for feature detection and image segmentation. In T. C. Fogarty (editor), *Evolutionary Computing*. Springer-Verlag, 1996.
- [11] Riccardo Poli. Genetic programming for image analysis. In John R. Koza, David E. Goldberg, David B. Fogel and Rick L. Riolo (editors), *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 363–368, Stanford University, CA, USA, 28-31, July 1996. MIT Press.
- [12] A. Song, V. Ciesielski and H.E Williams. Texture classifiers generated by genetic programming. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, Volume 1, pages 243–248, May 2002.
- [13] Andy Song, Thomas Loveard and Vic Ciesielski. Towards genetic programming for texture classification. *Lecture Notes in Computer Science*, Volume 2256, pages 461–472, 2001.
- [14] Walter Alden Tackett. Genetic generation of “dendritic” trees for image classification. In *Proceedings of WCNN93*, pages IV 646–649. IEEE Press, July 1993.
- [15] Mihran Tuceryan and Anil K. Jain. Texture analysis. In C. H. Chen, L. F. Pau and P. S. P. Wang (editors), *Handbook of Pattern Recognition and Computer Vision*, Chapter 2, pages 235–276. World Scientific, Singapore, 1993.
- [16] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco, Calif. : Morgan Kaufmann, 2000.
- [17] Mengjie Zhang and Victor Ciesielski. Genetic programming for multiple class object detection. In Norman Foo (editor), *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI'99)*, pages 180–192, Sydney, Australia, December 1999. Springer-Verlag Berlin Heidelberg. Lecture Notes in Artificial Intelligence (LNAI Volume 1747).