Texture Segmentation by Genetic Programming

Andy Song andy.song@rmit.edu.au
School of Computer Science and Information Technology, RMIT University,
PO Box 2476V, Melbourne Victoria, Australia

Vic Ciesielski vic.ciesielski@rmit.edu.au
School of Computer Science and Information Technology, RMIT University,
PO Box 2476V, Melbourne Victoria, Australia

Abstract

This paper describes a texture segmentation method using genetic programming (GP), which is one of the most powerful evolutionary computation algorithms. By choosing an appropriate representation texture, classifiers can be evolved without computing texture features. Due to the absence of time-consuming feature extraction, the evolved classifiers enable the development of the proposed texture segmentation algorithm. This GP based method can achieve a segmentation speed that is significantly higher than that of conventional methods. This method does not require a human expert to manually construct models for texture feature extraction. In an analysis of the evolved classifiers, it can be seen that these GP classifiers are not arbitrary. Certain textural regularities are captured by these classifiers to discriminate different textures. GP has been shown in this study as a feasible and a powerful approach for texture classification and segmentation, which are generally considered as complex vision tasks.

Keywords

Machine vision, image classification, texture classification, texture segmentation, genetic programming, evolutionary computation, machine learning.

1 Introduction

Texture is one of the most important visual clues. However, analyzing textures still remains an unsolved problem in machine vision. Texture analysis has two main aspects: classification and segmentation (Tuceryan and Jain, 1993). Texture segmentation, which can be considered as an extension of texture classification, is tackled by a novel approach in this study, using genetic programming (GP).

GP has been demonstrated to be a flexible method of problem solving for a diverse range of complex problems (Koza, 1999; Poli et al., 2008), such as classification. The evolutionary process implements a search for a good classifier. In previous investigations genetic programming techniques were shown to be capable of producing accurate classifiers in a variety of domains such as medical diagnosis (Guo and Nandi, 2006), object detection (Tackett, 1993; Howard et al., 2006), and image analysis (Chaudhry et al., 2007; Ross et al., 2005).

In this study we use GP to evolve texture classifiers and develop a texture segmentation method based on these evolved classifiers. The main goals of the study are

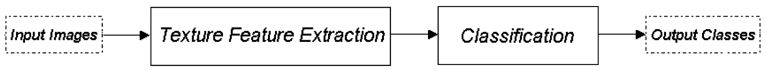


Figure 1: Conventional texture classification procedure.

- To investigate the use of GP in texture analysis without feature extraction.
- To develop a framework based on evolved classifiers for segmenting images containing two or more textures.
- To develop a fast texture segmentation method. Segmentation speed is essential for real world applications, especially real-time applications and applications in resource constrained environments.

In the next section, the background of texture analysis, and GP on image-related tasks, are discussed. Section 3 presents the evolution of texture classifiers without feature extraction. The use of the evolved classifiers for segmentation is presented in Section 4, which also includes a discussion on segmentation speed. Section 5 shows an analysis of evolved classifiers. The conclusion and discussion are in Section 6.

2 Background

2.1 Texture and Texture Analysis

Texture is usually understood as the characteristics of the appearance of a surface. The ability to recognize textures is essential to the success of many vision systems, either natural or artificial. However, there is no universally agreed definition of texture. Texture analysis mainly aims to computationally represent an intuitive perception of texture and to facilitate automatic processing of the texture information for artificial vision systems. The process of texture analysis usually produces some kind of numeric description of the texture. These are called texture features. The process of computing the texture features is known as feature extraction. Analysis of texture might provide primitives for texture definitions while the definitions can provide theoretical support for the analysis. Not surprisingly, just like the situation with texture definition, there is no universally accepted texture analysis method.[1]

2.1.1 Texture Classification

The objective of texture classification is to determine the category of a texture image (Varma and Zisserman, 2005). There are many applications of texture classification, such as satellite image classification (Coburn and Roberts, 2004), image retrieval (Howarth and Ruger, 2004), medical image classification (Rakebrandt et al., 2000), and industrial inspection (Maenpaa et al., 2003).

The conventional approach to accomplishing the texture classification task is shown in Figure 1 and is composed of two main steps. The first step is generating features of the classes of textures, usually by a particular texture feature extraction method such as the

[1]"The fact that the perception of texture has so many different dimensions is an important reason why there is no single method of texture representation that is adequate for a variety of textures" (Tuceryan and Jain, 1993).

co-occurrence matrix (Haralick et al., 1973), edgeness (Sonka et al., 1999), laws (Laws, 1980), run-length (Albregtsen et al., 2000), autocorrelation (Leu, 2001), Fourier transform (Chen and Chen, 1999; Tuceryan and Jain, 1993), Gabor transform (Grigorescu et al., 2002), Markov random fields (Chellappa and Chatterjee, 1985), and fractals (Sonka et al., 1999; Tuceryan and Jain, 1993). Once the features are available, a classification technique can be applied to discriminate textures from different categories. The performance of a texture classification task can be measured by classification accuracy.

2.1.2 Texture Segmentation

Texture segmentation is a more complex texture analysis task which is used to identify the boundaries between different textures. An advanced goal of texture segmentation is not only giving the boundaries between regions, but also identifying the texture in each region. Texture segmentation can be considered as an extension of texture classification in some cases if classification is used to determine the texture category of a pixel or a region. It has more significance in practice than texture classification because detecting boundaries and defining regions is highly desirable for machine vision systems. The applications of texture segmentation mainly aim to separate objects and background (Ozyildiz et al., 2002) or to identify different areas in a scene (Ruzon, 1997).

Texture segmentation can be also considered as a special case of image segmentation. It shares some of the principles of image segmentation such as partitioning based on inhomogeneity. However, typical image segmentation methods, such as edge detection, thresholding, and region growing, are not applicable to texture segmentation (Pratt, 1991).

2.2 Genetic Programming on Image-related Tasks

The use of GP on image related problems started soon after Koza (1992) introduced GP as a problem solving paradigm. Since 1993 a number of researchers have successfully adapted GP for various kinds of image related tasks, such as object detection (Tackett, 1993; Zhang, 2007), image classification (Ross et al., 2002, 2005; Wijesinghe and Ciesielski, 2007) and image processing tasks (Smith et al., 2005; Trujillo and Olague, 2006).

In general, these applications achieved satisfactory results using GP. Its advantage of not requiring domain knowledge has been widely recognized. In most of the investigations that involved other learning methods such as neural networks and decision trees, GP had superior performance, for example in object detection (Tackett, 1993; Zhang, 2000), image classification (Gray et al., 1996), and image segmentation (Poli, 1996).

The various studies of GP in image related areas are very different in terms of their domains and the data used. However, they approached the problems in similar ways.

- Representation. Most of the studies in this area use "canonical" program trees to represent solutions, although graph (Teller and Veloso, 1995) and linear (Harvey et al., 2002) representations have also been used.

- Supervised Learning. Most of the work used GP as a supervised learning method, although GP has shown to be capable of unsupervised learning by (De Falco et al., 2002). It is relatively easy to adapt GP to supervised learning since the fitness measure can be simply defined and evaluated as the differences between program outputs and predefined goals.

- Binary Problems. Most researchers represent their tasks as binary problems, for example object vs. non-object, face vs. non-face, text vs. picture, true vs. false, and so on. Multi-class problems can always be transformed into several binary problems. In binary problems, the natural boundary, zero, can be used as the boundary between two classes. This approach has been widely adopted by researchers. Positive output denotes object, face, text, correct class, and so on, while negative output denotes the opposite. One exception is the work of Zhang, who used multiple points for different classes of objects (Zhang, 2000).

- Function Set. The choice of function set largely depends on the task. Some studies only used simple arithmetic operators as functions. Some used a set of image processing functions (Roberts, 2003). The use of complex functions, such as sine, cosine, power, and specialized image progressing functions is not common in these studies. In character recognition tasks, and especially in classification tasks, the complex functions used are inherited from manually written algorithms, which are "not for the mere sake of solving the problem, but for increased interaction with human-written programs" (Andre, 1994, p. 467). It was actually found that very few successful individuals required all these functions.

- Terminal Set. It seems that there is also no consistent choice of terminal set. Some used pixels directly while others use extracted features including simple pixel statistics. Some work has investigated both approaches for the same tasks. The results suggest that there are no significant benefits of using complex features (Tackett, 1993; Howard and Roberts, 1999a,b). For example, Howard and Roberts found that using discrete Fourier transform (DFT) features did not offer much more help for their ship detection task (Howard and Roberts, 1999a,b) than raw pixels. Without features, GP was still able to provide satisfactory results. It has been hypothesized by Tackett that the good results might be due to GP "discovering inherent features which are better suited to the data than human-synthesized features" (Tackett, 1993, p. 308). In these works, a mixture of features and raw pixels were also used. The use of constant terminals can be found in the majority of these image-related GP applications, although there is also no consistent choice.

3 Evolving Texture Classifiers by Genetic Programming

Evolving texture classifiers is the first step toward segmentation. The purpose is generating classifiers that can reliably recognize whether a given cutout image contains a certain texture or not. Other than classification accuracy, classification speed is another important consideration because these classifiers will be used repetitively during the segmentation. A fast classifier is highly desirable. 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.

3.1 Single Step Approach

A diagram of the proposed single step approach is shown in Figure 2. In contrast, conventional texture classification methods need two major steps, feature extraction plus classification (see Figure 1). The inputs of a classifier are raw pixels rather than features. Although the approach is called the "single step" method, auxiliary processes such as image preprocessing and data preparation are also needed.

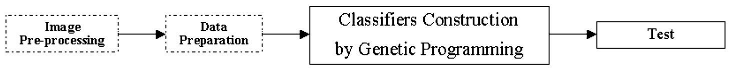


Figure 2: Diagram of single-step texture classification.

The major difference between the proposed approach and the conventional approach is that feature extraction is not required. The main reasons for considering the single step approach are:

- Unlike other classification methods, GP constructs classifiers by generating programs. This suggests that the generated classifiers could use pixel data to directly classify textures, since feature extraction and classification are also implemented as programs.
- In texture analysis, numerous feature extraction schemes have been proposed. However, none of the schemes is universally applicable due to the lack of knowledge of the nature of texture. In this situation, genetic programming might offer a new direction since it can evolve programs toward a solution without predefined domain knowledge.
- Designing a texture classification system is time-consuming. It usually involves selecting a good combination of "feature extraction + classification" and optimizing the chosen feature extractor in addition to the chosen classifier. It will be economically favorable if a texture classification program can be evolved automatically.
- Current texture feature extraction methods usually require considerable resources to compute the features before the classifier can be applied. Even if a classifier is fast, the overall process is still computationally expensive. This limits the application of such methods in some circumstances, such as real-time applications and applications in a small resource environment. In contrast, the programs evolved by genetic programming are relatively simple, small compared to feature extraction programs, and can be executed quickly. Only the creation of such a texture classifier will be resource intensive.

3.2 GP Configuration

Similar to most of the image-related GP work, we use program trees to represent a classifier. The class is determined by the tree output. If the output of a tree falls in a certain range, then the class assigned to that range is the classification result. The method for determining the class ranges is called dynamic range selection, because the ranges for class labels over a set of classes are determined dynamically (Loveard and Ciesielski, 2001). The dynamic range selection method of classifier representation was shown to be capable of producing accurate results over a variety of datasets in the medical and image processing domains (Loveard and Ciesielski 2001; Song et al., 2001).

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 choice of function and terminal sets is consistent with the studies discussed in Section 2.2.

Table 1: Function Set

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 2: Terminal Set

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

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:

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

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 dataset.

In our runs, a collection of data is used for training. The data consists of small cutouts, or subimages, sampled from the images of all classes. In the case of classifying one texture against another texture, about one thousand subimages are sampled from an image of the first texture to form "Class 1," and an equal number of subimages are sampled from an image of the other texture to form "Class 2." In the case of classifying one texture against a group of other textures, the one thousand subimages 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 textures, B, C, and D, one thousand subimages will be sampled from the image of texture A, and about 333 subimages will be sampled from the images of textures B, C, and D, respectively, to form "Class 2." The dataset will be split into training and test by a cross-validation process.

Our runs used a population size of 500 individuals. The initial population was generated by the ramp-half-and-half method. Roulette wheel selection was the method for filling up the mating pool. 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 an overall program depth limited to 17. Strongly typed GP was

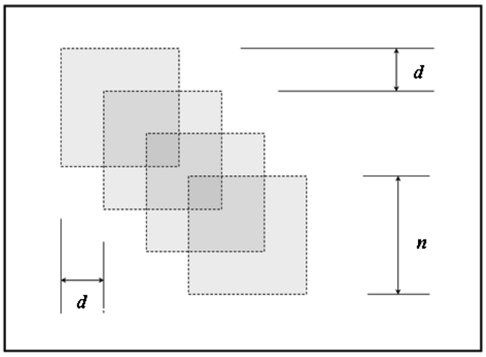


Figure 3: Illustration of the segmentation algorithm.

implemented to avoid generating illegal trees. The termination criteria for each run were either perfect classification—where classification accuracy was 100%—or after 50 generations had been processed. During the evolution 10-fold cross validation was also used to obtain validation accuracies of the evolved programs. The classifiers with highest training and validation accuracies were used for the later segmentation experiments. The evolutionary process might take up to 15 hours to complete on a Sun-Sparc workstation.

Our experiments used two kinds of textures: bitmap patterns and Brodatz textures. Bitmap patterns represent simple textures. These texture images are composed of pixels of only two values, **0** or **1**. Brodatz textures represent difficult textures. They are 8-bit grayscale images. Brodatz textures are labeled from D1 to D112, and originate from a photographic album published in 1966 (Brodatz, 1966). This album is now a kind of de facto standard database for texture-related research (Picard et al., 1993).

4 Texture Segmentation by Genetic Programming

4.1 Segmentation Algorithms

Texture segmentation can be approached by supervised learning or unsupervised learning. Our single-step method of classification is typical of supervised learning where the class information is available during training. Consequently the segmentation method based on the single-step approach is also supervised. In general there are two main categories of segmentation methods, region-based and boundary-based. The proposed method is a region-based approach since it is based on labeling regions rather than finding boundaries.

Figure 3 illustrates the basic algorithm which is described formally in Figure 4. The basic process uses a sweeping window of size $n \times n$ to sample an input image, where n is equal to the size of training images. At each window position, the sampled subimage will be classified by the GP program and all of its pixels will be marked accordingly. The distance between two adjacent window positions is d. The window sweeps an image row by row until it reaches the lower-right corner of the image. As shown in the figure, the window positions overlap. Therefore most of the pixels will have multiple class labels. To determine the final class of one pixel, a voting strategy is used. The class of the pixel is determined by majority voting.

```
Input:   $T_1, T_2$   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 $I$
         $n$          window size
         $d$          step size for moving window, $1 < d \leq n/2$
Output:  $O$          a two color image $O_{ij} = color1$ for texture 1 and
                      $O_{ij} = color2$ for texture 2

1. Generate a single-step classifier of which the input image size is $n \times n$.
2. Use the generated classifier to sweep the input image $I$.
    (a) Start at the top-left corner of the image.
    (b) Sample a subimage by an $n \times n$ 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 4: Segmentation algorithm for two textures.

The algorithm described in Figure 4 is specific to two texture problems and needs to be modified if there are more than two textures. This can be done in two ways: (1) The classifier in step 1 can be replaced by one that can discriminate the multiple textures or (2) multiple individual classifiers that discriminate texture T_i from all the others can be used in a voting mechanism. The second option is used because the evolved classifiers are mostly binary, to differentiate two classes of textures. It is possible to evolve multiclass classifiers, however this will usually result in less accurate classifiers. The segmentation algorithm that adopts this option is shown in Figure 5.

The combination of multiple classifiers shown in Figure 5 is similar to boosting, which considers the accuracy of each classifier. However, our method ranks classifiers based on their accuracy during cross-validation (based on Formula 1). The final decision is not made by the aggregation of classifiers with weights but by the one with the highest rank.

4.2 Segmentation Experiments

Based on the segmentation algorithms, experiments of segmenting two textures and of segmenting multiple textures have been conducted for both bitmap textures and Brodatz textures. The images used in segmentation were not used in GP training. However, due to the repetitive pattern of bitmap textures, there were duplicates between the training dataset and the subimages sampled from the test images.

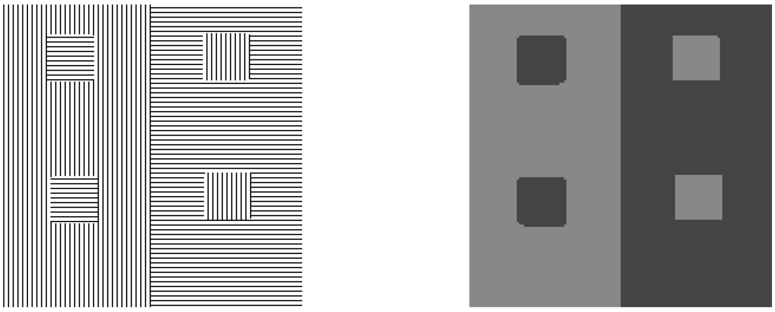
Input:	m	number of textures
	$T = \{T_1, T_2 \ldots T_m\}$	textures for which example images are available
	I	an image containing a number of textures $T', T' \subset T$
	w, h	width and height of I
	n	window size
	d	step size for moving window, $1 < d \leq n/2$
Output:	O	an image of m colors where each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

1. Generate m single-step classifiers of which the input image size is $n \times n$. 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 $Classifer_1$, $Classifier_2, \ldots, Classifier_m$.
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 subimage by an $n \times n$ 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 that only contains the regions.
      i. Label each pixel based on voting. If the majority of 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, that is, it is classified as Ti by $Classifieri$ and classified as T_j by $Classifier_j$, then the final decision will be made by the classifier which has the highest accuracy in cross validation. If $Classifier_i$ is more accurate than $Classifier_j$ on validation 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 5: Segmentation algorithm for multiple textures.

4.2.1 Bitmap Textures

Figure 6 shows an input image containing two bitmap textures and the segmented output image. Also shown are the parameter values used and the results. The overlap between two adjacent samples is an area of size 2×4. The texture regions in the output image are identified by two different intensity levels, gray and black. The training and validation accuracies in the table are the accuracies obtained during the generation of the classifier. The segmentation performance was evaluated by measuring the percentage of pixels that have been correctly labeled (see Formula 2). Performance measurement itself is an issue in segmentation. There are several approaches to evaluate



Two patterns with six regions, size 256×256 Output segmentation

Input Parameters		Results	
n	4	Training accuracy	100%
w	256	Validation accuracy	100%
h	256	CPU time (seconds)	0.07
d	2	Segmentation performance	99.83%

Figure 6: Segmenting two patterns with multiple regions.

the performance of a segmentation algorithm. In this study, the aim is to investigate a methodology for applying the single-step approach to segmentation rather than to compare it with other methods. Therefore the most straightforward measure was used to indicate segmentation performance.

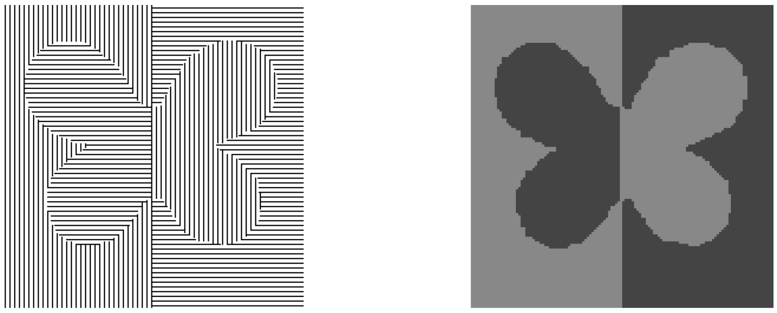
$$\textit{Segmentation performance} = \frac{\textit{Number of pixels being correctly labeled}}{\text{Total number of pixels in the image}} \times 100\% \qquad (2)$$

The region boundaries in the previous segmentation task are always straight, either horizontal or vertical. Such boundaries might be easier to handle because of the strictly square sampling window used. However, straight boundaries are rarely found in real life applications. Therefore, more variations of input images are introduced such as the input image in Figure 7. The segmented boundaries of the two butterfly wings are not as smooth as in the original image. Adjacent areas of two texture regions are usually problematic. However, the output boundaries closely resemble the actual ones, so the segmentation outcome is considered as very good.

4.2.2 Brodatz Textures

An image composed of four Brodatz textures is illustrated in Figure 8.[2] 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

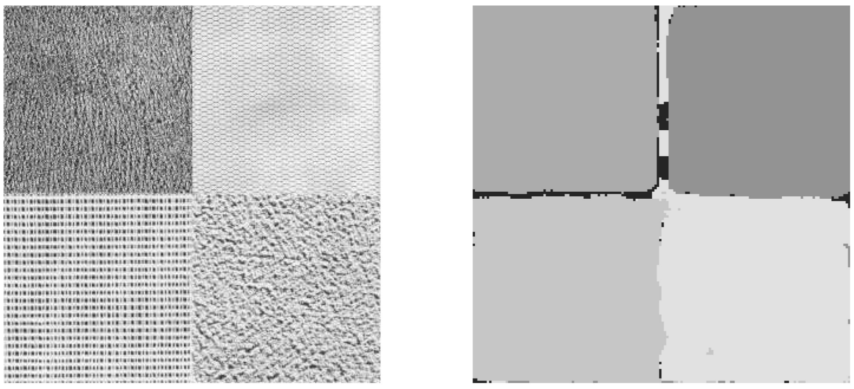
[2]Only fine and regular textures were selected. For coarse textures such as D10 and D30, the 16 × 16 window size would be too small to capture their textural characteristics.



Two patterns with complex region boundaries Output segmentation

CPU time (seconds)	0.07
Segmentation performance	98.06%

Figure 7: Segmenting two patterns with complex region boundaries.



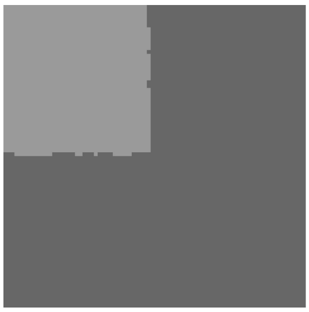
Original image, size 320 × 320 Segmentation output

Textures		Input parameters		Results	
Top-left	D24	n	16	Each color indicates one texture, black indicates the unidentified areas	
Top-right	D34	w	320		
Bottom-left	D21	h	320	CPU time (s)	2.58
Bottom-right	D57	d	2	Segmentation performance	97.48%

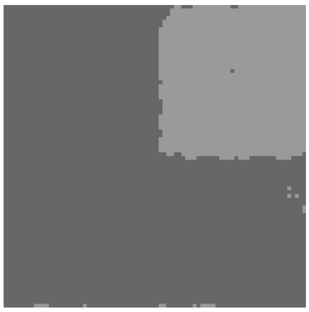
Figure 8: Segmenting four Brodatz textures, D21, D24, D34, and D57.

to boundaries that are known difficult areas. The input image size is 320 × 320. There were four classifiers participating in the segmentation.

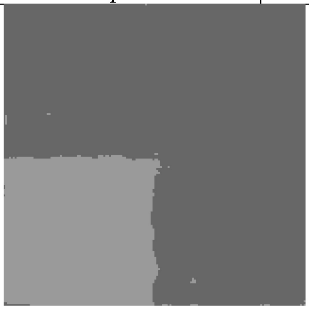
To assess the performance of the classifiers, each was applied to the segmentation task individually. The experimental results are presented in Figure 9, along with the training accuracy and validation accuracy of each classifier. The regions claimed by a classifier as "its own" texture are painted gray and regions not claimed by it are in 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



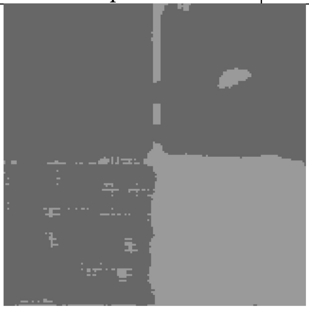
Results of D24 classifier	
Training accuracy	97.60%
Validation accuracy	96.00%
CPU time (s)	0.62
Segmentation performance	98.44%



Results of D34 classifier	
Training accuracy	94.00%
Validation accuracy	93.00%
CPU time (s)	0.49
Segmentation performance	99.45%



Results of D21 classifier	
Training accuracy	81.70%
Validation accuracy	82.00%
CPU time (s)	0.74
Segmentation performance	99.00%

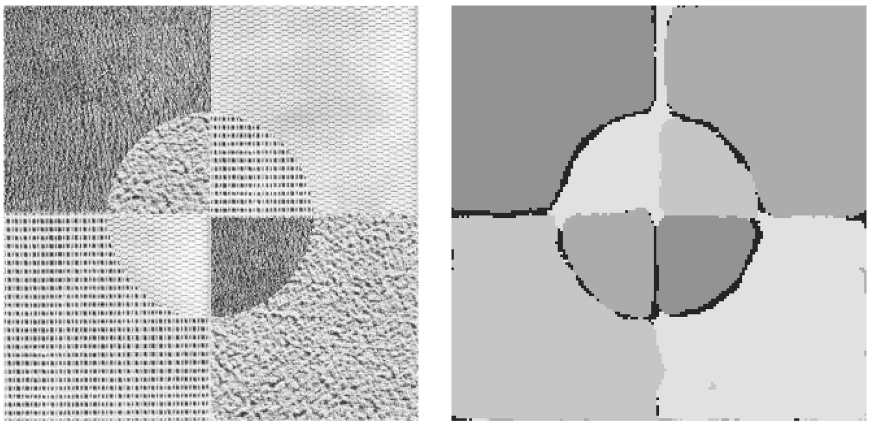


Results of D57 classifier	
Training accuracy	79.70%
Validation accuracy	78.00%
CPU time (s)	0.71
Segmentation performance	96.33%

Figure 9: Outputs of four classifiers.

poor training accuracy of only 79.70%. This classifier also marked some areas that 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 of their accuracy.

The CPU time spent by each classifier varies from 0.44 to 0.74 s. The variation is due to the different sizes of these classifiers. On average, each classifier spent about one quarter of the time used by the four classifiers together (2.58 s). This further illustrates that a single classifier is able to process a 320 $\times$ 320 image very quickly and adding one extra classifier will not significantly delay the segmentation. A variation of the four-texture image is designed to introduce both multiple regions and complex boundaries as shown in Figure 10. The image is similar to the four texture pattern, but has a circle at the center. The output reveals that the generated boundaries are close to the actual boundaries. Such results imply that the proposed method is able to handle complex boundaries even if multiple textures are involved. Moreover, it shows that multiple regions of the same texture can be identified.



Original image Segmentation output

Figure 10: Segmenting four textures with a circle.

4.3 Speed Comparison

The classifiers generated by the single-step approach have a simple structure. They have limited depth and contain no loops. So compared to conventional feature extraction algorithms, their computation costs are low. For example, the following classifier is the one for classifying D1 against all others. It has 668 nodes, of which 306 are functions. It contains 216 arithmetic operators ($+$, $-$, $*$, ., and /) and 90 logic operators (*if*, *between*, $<=$, $>=$, and $=$). The size and number of functions of other generated classifiers are similar.

```
(+ (* (if (n= (if (<= (/ PIXEL13 0.874945) (/ (- -0.593451 -0.620297) 0.110705)) (/ PIXEL911 PIXEL787) (+
PIXEL578 (+ PIXEL197 0.313259))) 0.797768) (* -0.308779 (+ (* 0.35431 PIXEL32) -0.164766)) (+ (* 0.0443723
(/ (* (if (Between 0.505077 (+ (+ (* 0.35431 PIXEL32) (+ -0.574929 -0.759044)) -0.965897) (- (if (<= (/ PIXEL13
0.874945) (/ PIXEL926 0.110705)) (+ (- (- (if (Between -0.360499 PIXEL898 0.875412) ... ...
```

This characteristic enables a fast voting strategy which can lead to much faster segmentation than the conventional "feature extraction + classification" approaches. The complexity of these generated classifiers is directly dependent on the number of operators, which is the number of functions in the program tree. The number of terminals is determined by the number of functions. Generally the classifiers created for classifying bitmap patterns are smaller than those for classifying Brodatz textures. No classifier for Brodatz textures was observed to have more than 700 functions.

In comparison, feature extraction algorithms are much more complex. Using Haralick features as an example, at least one 256×256 cooccurrence matrix needs to be generated in computing the features for an image with 256 levels of grayscale (Haralick et al., 1973). One of the simplest Haralick features, energy, is computed as

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

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

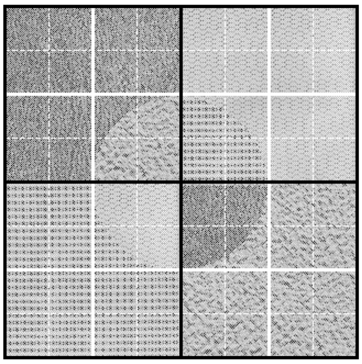


Figure 11: Three levels of splitting an input image.

To illustrate the above complexity comparison, the CPU times of using conventional methods were estimated. On the SUN Sparc workstation used for the experiments from Figures 6 to 10, 0.7 s is needed to compute Haralick features for a 16×16 grayscale image. Consider a hypothetical situation where a segmentation method has been developed based on the two-step approach and it uses an extremely fast classification method so the computation time for classification can be ignored. If the method uses a voting strategy and the step size of the moving window, d, is two, then the processing time for a 256×256 grayscale image is approximately

$$\left(\frac{256 - 16}{2} + 1\right)^2 \times 0.7 \approx 10{,}000 \text{ s}$$

The computational expense can be significantly reduced by adopting a split-and-merge approach and limiting the split to only three levels (as illustrated in Figure 11). In such a situation features are required to be computed on four 128×128 quarters for the first split, on 16 smaller quarters of size 64×64 for the second split and on 64 squares of size 32×32 for the third split. The times for computing Haralick features for images of size 128×128, 64×64, and 32×32 on the same SUN Sparc workstation are 4.58, 4.10, and 1.70 s, respectively. Therefore, the total CPU time for computing these features will be

$$4 \times 4.58 + 16 \times 4.10 + 64 \times 1.70 = 192.72 \text{ s}$$

If Gabor features are used, the CPU times for images of the above three sizes are 14.42, 3.53, and 1.08 s, respectively. The total time can then be estimated as

$$4 \times 14.42 + 16 \times 3.53 + 64 \times 1.08 = 183.28 \text{ s}$$

Clearly, whether the voting strategy or the split-and-merge strategy is used, the times taken for computing these features are already much longer than 2.58 s, the time for segmenting a 320×320 image by our proposed algorithm. Furthermore, three levels of split is not enough to produce a smooth boundary, as we showed in previous figures.

The above runtime estimates of a two-step approach are consistent with the literature. Jain and Kalle reported 122 s of processing time on a SUN Sparc-10 workstation to

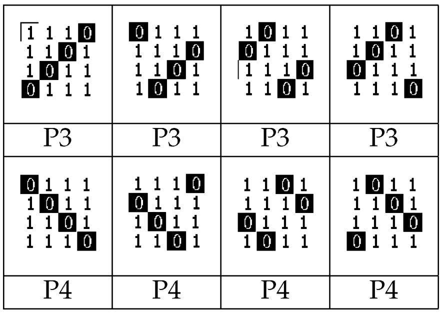


Figure 12: Subimages of P3 and P4.

segment a 256 × 256 grayscale image using Gabor filters and 109 s of processing time by their proposed method (Jain and Karu, 1996). Chang et al. evaluated different texture segmentation algorithms and reported the runtimes of "feature extraction plus classification" on a Sun Ultra Sparc. All of the times were more than 28 min (Chang et al., 1999). These runtimes reported in the literature are measured on different machines and under different conditions. They are not suitable for direct comparison. However, it still can be seen that intensive computation was required for feature extraction, which usually uses every pixel in an image and has quadratic complexity. In comparison, the evolved programs use only part of the pixels and have a linear complexity to tree size. The two step approach would require much more computation time than the single step approach.

5 Analysis of Evolved Texture Classifiers

Conventional texture analysis methods usually extract features that are based on hypotheses on the nature of texture or texture models. The single step approach does not have any prior assumption on the nature of texture; however, it can evolve classifiers. Is this success arbitrary? Have any texture characteristics or regularities been captured? To answer these questions, we analyzed the behaviors of these evolved GP classifiers.

Poor understandability of generated programs is a well-known problem of GP. It is often difficult to comprehend their behaviors. Therefore we simplified the analysis to bitmap textures and also incorporated a size penalty to encourage small classifiers. The modified fitness function is shown below. The high exponent on the accuracy is to emphasize classification performance, otherwise the evolution tends to generate programs as small as one node, but with bad accuracy.

$$f = \left(\frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TOTAL}}\right)^{14} \times 100 - (\mathit{Program\ Size}) \tag{3}$$

Two bitmap textures were used in the following investigation, forward diagonal P3 and backward diagonal P4. The task is to evolve a classifier to distinguish one from the other. Figure 12 shows enlarged 4 × 4 subimages of these two patterns.

The single step approach can easily produce classifiers with 100% accuracy on these two textures. About one third of them used only four pixels. Five of these classifers are

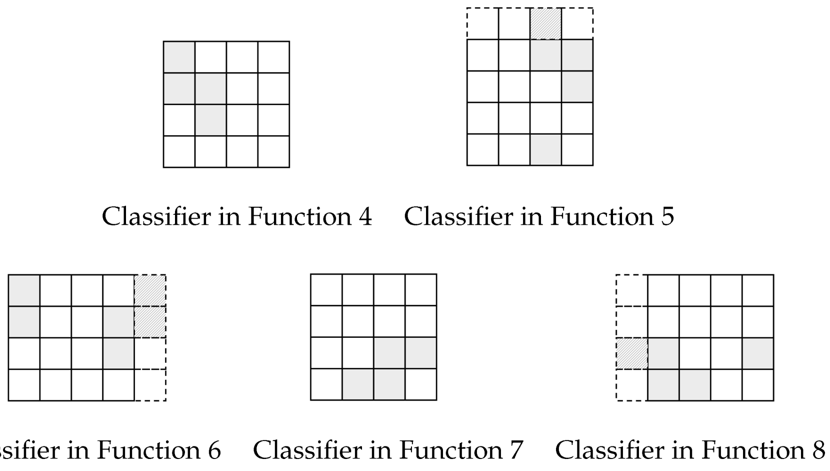


Figure 13: Five programs for classifying P3 and P4.

listed below and illustrated in Figure 13.

$$\begin{aligned} &Output = V(0,0) + V(1,0) + V(1,1) + V(2,1) \\ &Class\ is\ P3,\ \ if\, Output \in [\,3,4)\ \ Otherwise\ P4 \end{aligned} \tag{4}$$

$$\begin{aligned} &Output = V(0,2) + V(0,3) + V(1,3) + V(3,2) \\ &Class\ is\ P3,\ \ if\, Output \in [\,3,4)\ \ Otherwise P4 \end{aligned} \tag{5}$$

$$\begin{aligned} Output = V(0,0) + V(1,0) + V(1,3) + V(2,3) \\ Class\ is\ P3,\ \ if\, Output < 3\ or\ Output \geq 4\ \ Otherwise P4 \end{aligned} \tag{6}$$

$$\begin{aligned} Output = V(2,3) + V(2,2) + V(3,2) + V(3,1) \\ Class\ is\ P3,\ \ if\, Output < 3\ or\ Output \geq 4\ \ Otherwise\ P4 \end{aligned} \tag{7}$$

$$\begin{aligned} &Output = V(2,3) + V(2,0) + V(3,0) + V(3,1) \\ &Class\ is\ P3,\ \ if\, Output \in [\,3,4)\ \ Otherwise P4 \end{aligned} \tag{8}$$

These classifiers operate by adding the values of four selected pixels. The chosen pixels have a regular layout that is able to catch the characteristics of different patterns. For example, function 4 catches pixels (0,0), (1,0), (1,1), and (2,1); function 5 uses (0,2), (0,3), (1,3), and (3,2). As the patterns used in this investigation are all strictly regular, we know that the value of a pixel is equal to one 4 pixels away from it, in any direction. So we know that pixel (3,2) in a 4×4 grid is equivalent to $(-1,2)$. Therefore the two classifiers in Function 4 and 5 at the top of Figure 13 are equivalent and they can be generalized as a 4 pixel mask:

$$\begin{aligned} Output = V_{(y,x)} + V_{(y+1,x)} + V_{(y+1,x+1)} + V_{(y+2,x+1)} \\ Class\ is\ P3,\ \ if\, Output \in [\,3,4)\ Otherwise\ P4 \end{aligned} \tag{9}$$

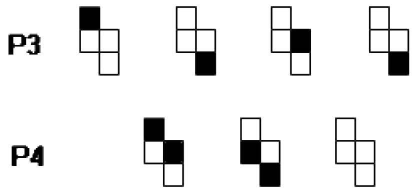


Figure 14: Possible inputs for function 9.

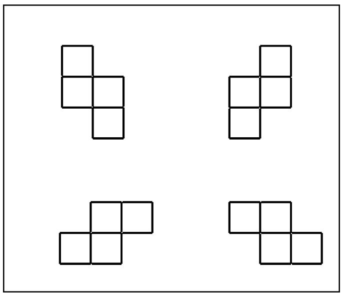


Figure 15: Four masks to differentiate P3 and P4.

Wherever this mask is placed over the pattern P3, there will be one and only one pixel with the value 0 (see Figure 14) in the mask, so the outputs for P3 subimages will always be 3. However applying the mask to the backward diagonal P4, either 2 black pixels or none will be captured.

Accordingly, the classifiers in functions 6, 7, and 8 can be generalized as well. These three masks together with the one shown in Function 9 are summarized in Figure 15. It can be seen that the "Z" shaped masks, encoded within the evolved programs, capture the regularity in the difference between P3 and P4. Although these masks have different orientations, they all work with the same mechanism, are small enough to be understood, yet capture enough information to tell the difference between P3 and P4.

A similar analysis has been performed on different pairs of bitmap textures as well as on the problem of one texture against a group of textures. Similar regularities have been revealed from these analyses. It can been shown that evolved classifiers by the single step approach behave as template matchers and frequency analyzers. They are able to capture regularities to differentiate textures.

6 Conclusion and Discussion

The work presented shows 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.

Analyzing classifiers on bitmap texture classification problems shows that the success of these classifiers is not ad hoc. Regularities of textures are captured. For the more complex problems—such as classifying gray level Brodatz textures—the GP generated classifiers are more difficult to understand, however we believe that certain underlying regularities are being captured. Furthermore, the analysis shows that GP as a population based problem solver is able to produce multiple texture classifiers for a single task. Using an ensemble of these classifiers could enhance the voting mechanism in the texture segmentation process.

The advantages and disadvantages of the described segmentation method are briefly summarized below. A more thorough investigation of this method is left for further work. Firstly the segmentation method is very fast. This makes it well suited for real time applications. Secondly this method can handle complex shapes and produce relatively smooth boundaries. To achieve that segmentation performance, the constituent classifiers do not have to be perfectly accurate. Additionally, different classifiers can be evolved in different runs and contribute to an ensemble which could enhance segmentation performance. Also the basic algorithm can readily be extended to segmentation of images with three or more textures.

However 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. Another issue is training time. 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 depending on the task. The method presented here is general, but the generated classifiers are problem-specific. For different applications different classifiers need to be retrained.

Other than presenting a GP-based texture segmentation method, this study shows GP as a powerful problem solving mechanism, even in a difficult domain like texture analysis. It is a viable alternative to tackle an unsolved complex problem, especially in vision. This work opens up many interesting topics for further exploration, such as how to more efficiently utilize an ensemble of classifiers, whether it is possible to address unsupervised texture segmentation by GP since a real world system is likely to encounter unseen textures, and how to generalize the segmentation methodology so it could work with non-texture segmentation methods as well.

References

Albregtsen, F., Nielsen, B., and Danielsen, H. (2000). Adaptive gray level run length features from class distance matrices. In *Proceedings of the 15th International Conference on Pattern Recognition*, Vol. 3 (pp. 738–741).

Andre, D. (1994). Learning and upgrading rules for an OCR system using genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*.

Brodatz, P. (1966). *Textures: A photographic album for artists and designers*. New York: Dover.

Chang, K., Bowyer, K., and Sivagurunath, M. (1999). Evaluation of texture segmentation algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1999*, Vol. 1 (pp. 294–299).

Chaudhry, A., Khan, A., Ali, A., and Mirza, A. M. (2007). A hybrid image restoration approach: Using fuzzy punctual kriging and genetic programming. *International Journal of Imaging Systems and Technology* 17(4), 224–231.

Chellappa, R., and Chatterjee, S. (1985). Classification of textures using Gaussian Markov random fields. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 33(4), 959–963.

Chen, C.-C., and Chen, C.-C. (1999). Filtering methods for texture discrimination. *Pattern Recognition Letters* 20(8), 783–790.

Coburn, C., and Roberts, A. C. B. (2004). A multiscale texture analysis procedure for improved forest stand classification. *International Journal of Remote Sensing* 25(20), 4287–4308.

De Falco, I., Tarantino, E., and Della Cioppa, A. (2002). Unsupervised spectral pattern recognition for multispectral images by means of a genetic programming approach. In *Proceedings of the 2002 Congress on Evolutionary Computation, 2002* (pp. 231–236).

Gray, H. F., Maxwell, R. J., Martinez-Perez, I., Arus, C., and Cerdan, S. (1996). Genetic programming for classification of brain tumors from nuclear magnetic resonance biopsy spectra. In J. R. Koza et al. (Eds.), *Genetic Programming 1996: Proceedings of the First Annual Conference* (p. 424).

Grigorescu, S., Petkov, N., and Kruizinga, P. (2002). Comparison of texture features based on Gabor filters. *IEEE Transactions on Image Processing 11*(10), 1160–1167.

Guo, H., and Nandi, A. K. (2006). Breast cancer diagnosis using genetic programming generated feature. *Pattern Recognition* 39(5), 980–987.

Haralick, R. M., Shanmugam, K., and Dinstein, I. (1973). Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics SMC-3*(6), 610–621.

Harvey, N., Theiler, J., Brumby, S., Perkins, S., Szymanski, J., and Bloch, J. (2002). Comparison of genie and conventional supervised classifiers for multispectral image feature extraction. *IEEE Transactions on Geoscience and Remote Sensing* 40(2), 393–404.

Howard, D., and Roberts, S. (1999a). Object detection by multiple textural analyzers. In *Proceedings of the 1999 Congress on Evolutionary Computation*, Vol. 2 (pp. 850–854).

Howard, D., and Roberts, S. C. (1999b). Evolving object detectors for infrared imagery: A comparison of texture analysis against simple statistics. In K. Miettinen, P. Neittaanmäki, M. M. Mäkelä, and J. Perriaux (Eds.), *Evolutionary algorithms in engineering and computer science*. Hoboken, NJ: Wiley.

Howard, D., Roberts, S. C., and Ryan, C. (2006). Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance. *Pattern Recognition Letters 27*(11), 1275–1288.

Howarth, P., and Ruger, S. (2004). Evaluation of texture features for content-based image retrieval. In *Proceedings of Third International Conference on Image and Video Retrieval*, Vol. 3115 of *Lecture Notes in Computer Science* (pp. 326–334).

Jain, A. K., and Karu, K. (1996). Learning texture discrimination masks. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 195–205.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Koza, J. R. (1999). *Genetic programming III: Darwinian invention and problem solving*. San Francisco: Morgan Kaufmann.

Laws, K. I. (1980). *Textured image segmentation*. Ph. D. thesis, University of Southern California, Los Angeles.

Leu, J.-G. (2001). On indexing the periodicity of image textures. *Image and Vision Computing* 19(13), 987–1000.

Loveard, T. and Ciesielski, V. (2001). Representing classification problems in genetic programming. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001* (pp. 27–30).

Maenpaa, T., Viertola, J., and Pietikainen, M. (2003). Optimising colour and texture features for real-time visual inspection. *Pattern Analysis and Applications* 6(3), 169–175.

Ozyildiz, E., Krahnstver, N., and Sharma, R. (2002). Adaptive texture and color segmentation for tracking moving objects. *Pattern Recognition 35*(10), 2013–2029.

Picard, R., Kabir, T., and Liu, F. (1993). Real-time recognition with the entire Brodatz texture database. In *CVPR '93, 1993 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 638–639).

Poli, R. (1996). Genetic programming for image analysis. In J. R. Koza et al. (Eds.), *Genetic Programming 1996: Proceedings of the First Annual Conference* (pp. 363–368).

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk. (With contributions by J. R. Koza.)

Pratt, W. K. (1991). *Digital image processing* (2nd ed.), Chap. 17 (pp. 557–596). New York: Wiley.

Rakebrandt, F., Crawford, D. C., Havard, D. C. D., and Woodcock, J. P. (2000). Relationship between ultrasound texture classification images and histology of atherosclerotic plaque. *Ultrasound in Medicine and Biology 26*(9), 1393–1402.

Roberts, M. E. (2003). The effectiveness of cost based subtree caching mechanisms in typed genetic programming for image segmentation. In G. R. Raidl et al. (Eds.), *Applications of Evolutionary Computing, EvoWorkshops2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM*, Vol. 2611 of *Lecture Notes in Computer Science* (pp. 444–454).

Ross, B. J., Gualtieri, A. G., Fueten, F., and Budkewitsch, P. (2002). Hyperspectral image analysis using genetic programming. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*.

Ross, B. J., Gualtieri, A. G., Fueten, F., and Budkewitsch, P. (2005). Hyperspectral image analysis using genetic programming. *Applied Soft Computing* 5(2), 147–156.

Ruzon, M. A. (1997). Texture segmentation on Mars. Available at http://robotics.stanford .edu/~ruzon/NASA/.

Smith, S. L., Leggett, S., and Tyrrell, A. M. (2005). An implicit context representation for evolving image processing filters. In *Applications of Evolutionary Computing, EvoWorkshops2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*.

Song, A., Loveard, T., and Ciesielski, V. (2001). Towards genetic programming for texture classification. *Lecture Notes in Computer Science* 2256, 461–472.

Sonka, M., Hlavac, V., and Boyle, R. (1999). *Image Processing, Analysis, and Machine Vision* (2nd ed.). Pacific Grove, CA: PWS.

Tackett, W. A. (1993). Genetic programming for feature discovery and image discrimination. In S. Forrest (Ed.), *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93* (pp. 303–309).

Teller, A., and Veloso, M. (1995). PADO: Learning tree structured algorithms for orchestration into an object recognition system. Technical Report CMU-CS-95-101, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

Trujillo, L., and Olague, G. (2006). Synthesis of interest point detectors through genetic programming. In *GECCO 2006: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*.

Tuceryan, M., and Jain, A. K. (1993). Texture analysis. In C. H. Chen, L. F. Pau, and P. S. P. Wang (Eds.), *Handbook of Pattern Recognition and Computer Vision*, Chap. 2 (pp. 235–276). Singapore: World Scientific.

Varma, M., and Zisserman, A. (2005). A statistical approach to texture classification from single images. *International Journal of Computer Vision* 62(1–2), 61–81.

Wijesinghe, G., and Ciesielski, V. (2007). Using restricted loops in genetic programming for image classification. In D. Srinivasan and L. Wang (Eds.), *2007 IEEE Congress on Evolutionary Computation* (pp. 4569–4576).

Zhang, M. (2000). *A domain independent approach to 2D object detection based on the neural and genetic paradigms*. Ph. D. thesis, Royal Melbourne Institute of Technology.

Zhang, M. (2007). Improving object detection performance with genetic programming. *International Journal on Artificial Intelligence Tools* 16(5), 849–873.