Chapter 12

Genetic Programming based Texture Filtering Framework

Mario Köppen, Bertram Nickolay

Fraunhofer IPK Berlin
Department Pattern Recognition
Pascalstr. 8–9, 10587 Berlin, Germany

Abstract

A framework is presented, which allows for the automated generation of texture filters by exploiting the 2D-Lookup algorithm and its optimization by evolutionary algorithms. To use the framework, the user has to give an original image, containing the structural property-of-interest (e.g. a surface fault), and a binary image (goal image), wherein each position of the structural property-of-interest is labeled with the foreground color. Doing so, the framework becomes capable of evolving the configuration of the 2D-Lookup algorithm towards a texture filter for the structural property-of-interest. Genetic programming (GP) is used as the evolutionary algorithm. For this GP approach, a filter generator derives two operations based on formal superoperators from the tree, which represents an individual of the evolving population. The specification of the 2D-Lookup matrix is performed by a relaxation technique. The approach will be demonstrated on texture fault examples.

Keywords : pattern recognition, scene analysis, image processing, texture analysis, texture filtering, mathematical morphology, 2D-Lookup algorithm, evolutionary algorithms, genetic algorithms, genetic programming, fitness function, relaxation, convolution, ordered weighted averaging, texture numbers, ordered weighted minimum, multilayer backpropagation neural network, crossover, mutation, document preprocessing, visual inspection of surfaces, handwriting extraction, image processing framework

12.1 Introduction

Textures are homogeneous visual patterns that we find in natural or synthetic scenes. They are made of local micropatterns, repeated somehow,

producing the sensation of uniformity. Texture perception plays an important role in human vision. It is used to detect and distinguish objects, to infer surface orientation and perspective, and to determine shape in 3D scenes. An interesting psychological observation is the fact that human beings are not able to describe textures clearly and objectively, but only subjectively by using a fuzzy characterization of them [**?**].

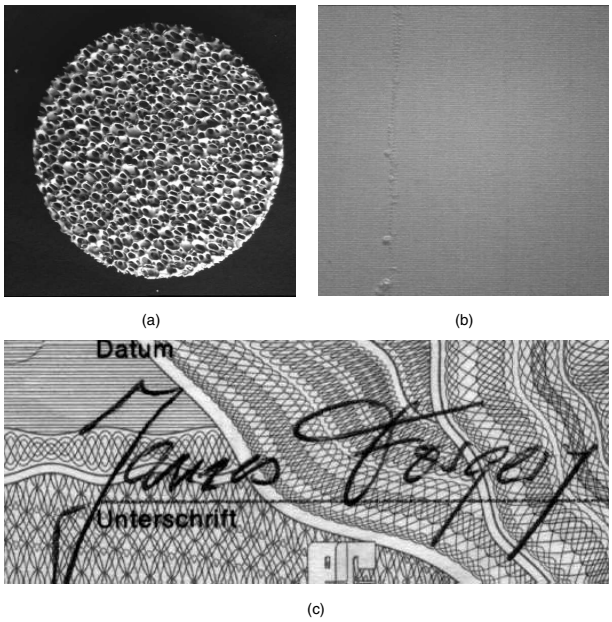

Fig. 12.1 Textures: ceramic filter (a), textile fault (b) and handwriting on bankcheck background texture (c).

Texture analysis is a subfield of image processing, which is concerned with the exploitation of pictorial textures for various image inspection tasks (see fig. 12.1). Such tasks, typically handled by methods of texture analysis, are texture segmentation, texture fault detection, texture synthesis and texture removal.

The purpose of texture filtering is to find image processing operations based on pixel information, which assign a texture class to a subregion of the image. Usually, the pixels, whose grayvalues or color values are used

for these computations*, are taken from the image region, which has to be classified. Typical regions are: local neighborhoods (i.e., a pixel and its direct neighbours); regions-of-interest (connected subsets of the set of all pixels); windows (rectangularily bounded subsets); or the whole image. Means for computations are based on a rich variability of mathematical or information theoretical concepts, as, e.g., statistical approaches, structural approaches, fractal approaches or spectral approaches [**?**].

A given texture filtering approach can be considered a framework. This means, that some components of the processing chain and the relations between them remain open for adaptation, i.e. the texture filter has to be *configured* (by numerical or structural parameters, by operator selection and so forth) in order to fulfill its task. In some cases, the configuration might become too complex to be supplied by an user of the superposed system, of which the framework is a part. Adaptive techniques are needed in this case.

Since the early days of computer vision, the feature based classification approach has become the primary texture analysis technique for the treatment of images of textured surfaces. The key steps of the feature classification approach are: image acquisition, image preprocessing, feature extraction, feature selection and feature classification [**?**], [**?**]. The feature classification approach can be employed as a texture filter as well. Its main disadvantage is the underlying concept of a single processing chain, which is "as strong as its weakest part." In order to improve reliability and robustness of the texture filters, backtracking in the processing chain may be used. However, this is impractical, since the effect of modification of a part of the chain at its end can hardly be predicted, and since the number of alternative settings to explore in order to find better ones grows exponentially with the number of alternatives for each part. A better idea is to decompose the processing flow into several parallel parts with a knowledge of what has to be acquired by these subparts at its ends (for example that an edge image should be the result). Finally, the subparts are fused by a fixed algorithm. An example for this is the fusion of an edge image and a pre-classified image by the watershed transformation [**?**]. If there are exactly two subparts, the approach is referred to as 2D framework [**?**].

This paper presents a special 2D texture filtering framework based on the so-called 2D-Lookup, and its configuration by means of evolutionary

*These values are referred to as pixel values in the following.

algorithms. The 2D-Lookup framework allows, by its configuration, to represent a very large number of texture filters. By genetic programming (GP), framework configurations are evolved, which meet the user-given filtering goal as good as possible. This paper is organized as follows: in section 12.2, the framework LUCIFER2[†] is described. Its subsections detail the 2D-Lookup algorithm, recall evolutionary algorithms, give the fitness measure used and the relaxation procedure for deriving a 2D-Lookup matrix from two operation images and detail the structure of a tree, which represents an individual of a GP. Then, in section **??**, this framework is applied to a set of texture fault detection tasks. The paper ends with the summary, an acknowledgment and the reference.

12.2 The LUCIFER2 Framework

The purpose of the presented framework is to design texture filters. Trained by user-provided examples, the adapted filters are able to separate a textured background from a foreground structure. Possible applications for these texture filters are: texture fault detection, texture border detection or handwriting extraction (on a bankcheck with textured background). These problems typically arise in fields like visual surface inspection on fabrics or optical document preprocessing.

The framework (see fig. 12.2) is composed of (user- supplied) original image, filter generator, operation images 1 and 2, result image, (user-supplied) goal image, 2D-Lookup matrix, comparing unit and filter generation signal.

The framework can be thought of as being composed of three (overlapping) layers.

(1) The instruction layer, which consists of the user-supplied parts of the framework: original image and goal image. The user may also supply other components (operation 1, operation 2, 2D-Lookup matrix), for maintenance purposes.

(2) The algorithm layer performs the actual 2D-Lookup, once all of its components (original image, operation 1, operation 2 and 2D-Lookup matrix) are given.

(3) The adaptation layer contains all adaptable components of the frame-

[†]This is an acronym for Lookup Compositional Inference System. Number 2 indicates that there was a genetic algorithm based version 1 [**?**].

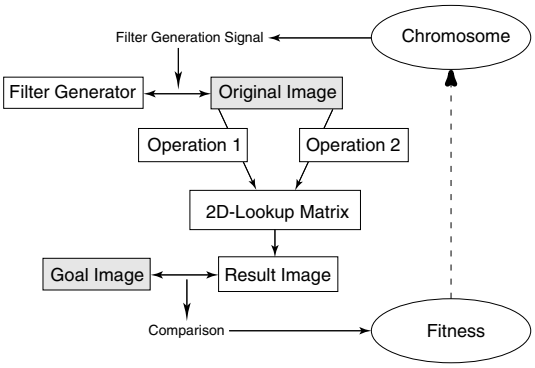

Fig. 12.2 The Framework for 2D-Lookup based texture filter generation.

work (operation 1, operation 2, 2D-Lookup matrix) and additional components for performing the adaptation (comparison unit, filter generator).

For the instruction layer, the user interface has been designed as simple as possible. The user instructs LUCIFER2 by manually drawing a (binary) goal image from the original image (by a photo retouching program as Photoshop). In this image, texture background is set to White and texture foreground (e.g. the texture fault, handwriting on a textured bankcheck background) to Black (see Fig.12.3 for an example). Rest of the approach is data-driven. No special texture model has to be known by the user. There are no further requirements for the goal image.

The algorithm layer performs the 2D-Lookup algorithm, which will be described in the next subsection. The algorithm decomposes the filter operation into a set of partial steps, each of which might be adapted to meet the user's instruction.

Adaptation is considered an optimization problem, and evolutionary algorithms are used for performing this adaptation. The fitness function is computed with the degree of resemblance between result image of an individual-specified 2D-Lookup and the goal image.

12.2.1 *2D-Lookup Algorithm*

The 2D-Lookup algorithm stems from mathematical morphology [**?**], [**?**]. It was primarily intended for the segmentation of color images. However,

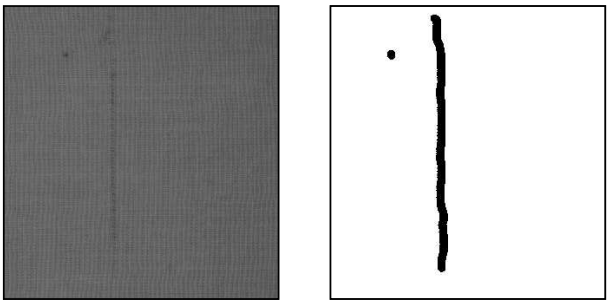

Fig. 12.3 Example for texture image containing fault (a) and goal image, as given by the user (b).

the algorithm can be generalized to use for grayvalue images as well.

For starting off the 2D-Lookup algorithm, the two operation images 1 and 2, which are of equal size, need to be provided. This is achieved by the filter generation signal, which is under the control of the individuals of the evolving population. The filter generation signal causes the filter generator to determine two image processing operations, which are applied to the original image. The 2D-Lookup algorithm goes over all common positions of the two operation images. For each position, the two pixel values at this position in operation images 1 and 2 are used as indices for looking-up the 2D-Lookup matrix. The matrix element, which is found there, is used as pixel value for this position of the result image. If the matrix is bi-valued (as for the goal image), the resultant image is a binary image.

Let I_1 and I_2 be two grayvalue images, defined by their image functions g_1 and g_2 over their common domain $P \subseteq \mathcal{N} \times \mathcal{N}$:

$$\begin{aligned} g_1 : &\quad P \rightarrow \{0, \ldots, g_{max}\} \\ g_2 : &\quad P \rightarrow \{0, \ldots, g_{max}\} \end{aligned} \tag{1}$$

The 2D-Lookup matrix is also given as an image function l, but its domain is not the set of all image positions but the set of tupels of possible grayvalue pairs $\{0, \ldots, g_{max}\} \times \{0, \ldots, g_{max}\}$,

$$l : \quad \{0, \ldots, g_{max}\} \times \{0, \ldots, g_{max}\} \rightarrow S \subseteq \{0, \ldots, g_{max}\}. \tag{2}$$

Then, the resultant image function is given by:

$$\begin{aligned} r &: P \to S \\ r(x,y) &= l(g_1(x,y), g_2(x,y)). \end{aligned} \tag{3}$$

In standard applications, every grayvalue is coded by eight bit, resulting in a maximum grayvalue of 255. Also, the domain of the image function is a rectangle. In this case, the 2D-Lookup is performed by the following (object-oriented) pseudo-code:

```
for x=0 to img width -1 do
begin
   for y=0 to img height-1 do
   begin
      g1 = g1(x,y)
      g2 = g2(x,y)
      out(x,y) = l(g1,g2)
   end y
end x
```

To give a simple example for the 2D-Lookup procedure, $g_{max} = 3$ is assumed in the following. Let

g_1 :

0	1	2
0	3	3

and g_2 :

2	3	1
2	3	2

be the two input images and the 2D-Lookup matrix be given by

l :

g_2 \ g_1	0	1	2	3
0	0	0	1	1
1	0	1	2	2
2	1	2	3	3
3	2	3	3	2

Then, the resultant image is

r :

$l(0,2)$	$l(1,3)$	$l(2,1)$
$l(0,2)$	$l(3,3)$	$l(3,2)$

$=$

1	3	2
1	2	3

Since the goal image is supplied as a binary one and in order to keep user instruction as simple as possible, in the following the 2D-Lookup matrix contains only binary entries Black (0) and White (1).

12.2.2 *Genetic Algorithms and Genetic Programming*

Evolutionary algorithms are a family of computer models based on the mechanics of natural selection and natural genetics. Among them are genetic algorithms (GA) [**?**] and genetic programming (GP) [**?**]. Genetic algorithms were introduced and investigated by John Holland [**?**]. Later, they became popular by the book of David Goldberg [**?**]. Also, consider the GA tutorial of David Whitley [**?**] as a very good introduction to the field.

GAs and GPs are typically used for optimization problems. An optimization problem is given by a mapping $F: X \to Y$. The task is to find an element $x \in X$ for which $y = f(x), y \in Y$ is optimal in some sense. Genetic algorithms encodes a potential solution on a simple chromosome-like data structure, and apply genetic operators such as crossover or mutation to these structures. Then, the potential solution is decoded to the value x in the *search space* X, and $y = f(x)$ is computed. The obtained value y is considered as a quality measure, i.e. the *fitness* for this data structure. Some genetic operators, such as the mating selection, are under control of these fitness values, some other, like the mutation, are not related to fitness at all.

An implementation of a GA begins with a population of "chromosomes" (generation 1). For standard GA, each chromosome (also referred to as individual) is represented as a bitstring of a fixed length (e.g. `0101101` as a bitstring of length 7). Then, the genetic operators are applied onto all bitstrings iteratively in a fixed order, going from one generation to the next until a given goal (e.g. fitness value exceeds a given threshold or a predefined number of generations was completed) is met. Finally, the individual (or chromosome) with the best fitness value in the final generation is taken as the evolved solution of the optimization problem.

Figure 12.4 illustrates the iteration of a GA from generation n to generation $(n+1)$. At first, $2m$ bitstrings are selected out of the k individuals of generation n for mating. Usually, this is done by fitness-proportionate selection, i.e., the relative probability for an individual to be selected is proportional to its fitness value. The better the fitness, the better is the chance to spread out its "genetic material" (i.e., some of its bits) over the next generation.

Once the $2m$ individuals are chosen, they are paired. In the two bitstrings of each pair, a common splitting point is randomly selected, and a new bitstring is constructed by combining a half of the first bitstring with

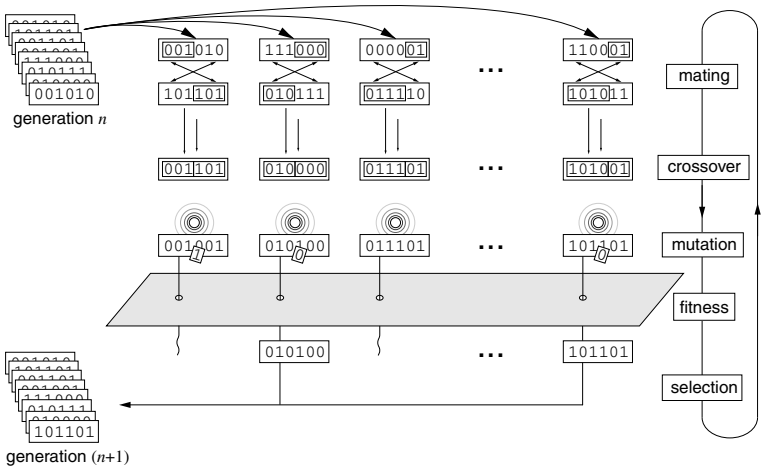

Fig. 12.4 General flow of a genetic algorithm: from generation n, individuals are selected for mating, according to their fitness. By crossover, new individuals are generated from the selected ones. Further, these new individuals are mutated, and its fitness function is computed. Only the individuals with the highest fitness values constitute the next generation $(n + 1)$.

the other half of the other bitstring. Then, the new individuals are mutated, i.e. some of its bits are reversed with a given (usually small) probability. This gives the so-called m *children* of parent generation n.

Now, the fitness values of the children are evaluated by decoding them into x values and computing the $f(x)$. Some of the children might have a better fitness than its parents. From the k individuals of generation n and the m children, the best k individuals constitute the next generation $(n + 1)$.

While randomized, GAs are no simple random walks. For the standard GA, John Holland has derived the well-known Schemata Theorem, which models a GA by means of the so-called schematas (or similarity templates). A schema is an incomplete bitstring in the sense that it contains unspecified bits. An example for a schema is `10*110`, which leaves position 3 unspecified. `101110` is a realization of this schema. Generation n contains each possible schema to some extent. It can be said, that such a schema is tested by the GA, or that trials are allocated to it by the GA. Now, one measure for a schema is the average fitness of all of its realizations. A second measure is the ratio of this avarage to the "average average" of

all schemata present in the generation n, i.e. its *above-averageness*. The Schemata Theorem relates the rate of a schema within a population with this measure. It says, that the rate of a schema within a population grows exponentially with its above-averageness. The most important point here is that all schematas are tested in parallel.

Strongly related to the application of a GA is the encoding problem. In general, GAs are applied to highly non-linear, complex problems, where it is hard to find a model which provides an approach to the solution. In these applications, they are the most simple approach. However, a GA is not guaranteed to find the global optimum of a problem. It only ensures, by the Schemata Theorem, to find better solutions than the random initialized ones. GAs find evolved solutions.

Genetic programming, as introduced by John Koza [**?**], is in some essential points very similar to a GA: there are generations, genetic operators as crossover and mutation, and there is a fitness function. However, the GP does not evolve bitstrings or other fixed data structures but it evolves *computer programs*. The computer programs are represented by its grammatical trees (shortly: trees). Each tree is composed of the available simple programming ingredients (nodes and terminals). The fitness of a tree is obtained by *performing* the tree as a computer program. GP then proceeds with iteratively applying the standard genetic operations, selection and crossover. For crossover, two individuals are chosen randomly, but according to their fitness. Then, randomly chosen subtrees are swapped. This gives two new children individuals from its two parent individuals. Based on the fitness values of the newly created children, they may become members of the next generation.

For applying GP, there is no encoding problem. However, the specification of suitable node and terminal functions might be as complicated as the specification of an encoding procedure. The question, whether the Schemata Theorem applies to GP search as well, is still discussed. The problem here are redundancies within the trees, leading to solutions with equal fitness, but different structure. It was shown by William Langdon and Riccardo Poli [**?**], that this growth in redundancy is caused by the fitness pressure itself (the effect is referred to as fitness-bloating). From this, there are two hints for succesfully applying GP: don't use simple node functions, and keep the trees small in its depth.

Since the upcoming of Soft Computing, there has been a huge amount of proposals about variations and modifications of GAs and GPs. Also,

many hybrid systems, combining evolutionary algorithms with neural networks and/or fuzzy logic, has been proposed. Some new families of evolutionary algorithms appeared, too (as cultural algorithms, ant algorithms, scout algorithms, immune algorithms). Also, modified GAs were successfully applied to hard-to-handle problem fields like multiobjective optimization problems. It is far beyond the scope of this paper, to give even an overview of these proposals. The inspiration from nature and from living systems has helped to produce this new challenge in the design of powerful and versatile search algorithms.

In the following, GP is just regarded as search technique for 2D-Lookup algorithm configuration. A recently presented GA approach [**?**] was proven to be outperformed by the GP approach, which is detailed here. In the following subsections it is described, how the fitness of a binary result image and the goal image is computed, how this fitness measure is reused for deriving an optimal 2D-Lookup matrix from two operation images, and how the trees of a GP population are built up.

12.2.3 *Fitness Function*

In order to compare the output image of the 2D- Lookup with the goal image, a quality function has to be designed for the comparison of two binary images. First, the definition of this fitness function will be given, then it will be discussed.

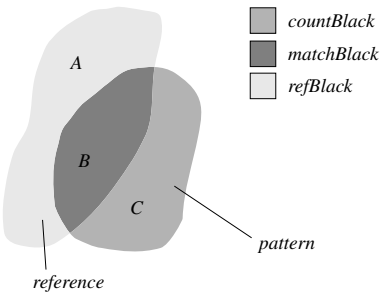

Fig. 12.5 Terms for fitness evaluation.

Consider figure 12.5, where two sets are shown, the reference set of the goal image and the pattern set of the result image.

Therein, *countBlack* is the number of black pixels in the result image

$(B + C)$, $matchBlack$ is the number of black pixels of the result image, which are also black in the goal image (B), and $refBlack$ is the number of black pixels of the goal image $(A + B)$. Then, the following ratios can be computed:

$$r_1 = \frac{matchBlack}{refBlack} \tag{4}$$

is the amount of reference pixels matched by the pattern,

$$r_2 = 1.0 - \frac{countBlack - matchBlack}{N - refBlack} \tag{5}$$

is the amount of correct white pixels set in the result image (N is the total number of image pixels), and

$$r_3 = \frac{matchBlack}{countBlack} \tag{6}$$

is the amount of matching pixels of the result image. The multiple objective here is to increase these measures simultaneously. After performing some experiments with the framework, it was decided to use the following weighted sum of these three objectives as fitness measure:

$$f = 0.1r_1 + 0.5r_2 + 0.4r_3 \tag{7}$$

This fitness measure has the following properties:

(1) It counts better for subsets of the reference. Subsets obtain a fitness value of at least 0.9, since r_2 and r_3 are 1 in this case.
(2) It counts better for subsets of the reference, which are supersets of other subsets of the reference.
(3) A white image gives a fitness of 0.5, therewith refusing to assign a good fitness value to the empty subset of the reference.

These properties make this fitness measure useful for genetic search. A genetic search evolves its population towards the higher weighted objective first. In our case this means, that measure r_2, weighted with 0.5, is evolved first. In other words, the first subgoal of the genetic search is to allocate as many correct white positions as possible. Due to the weighting of 0.4 for the r_3-part, the search then tries to allocate correct black positions of the reference, while the correct white allocations persist in the pattern. Once the pattern is reduced to a subset of the reference, the only way to increase

the fitness is to expand the subset towards the whole reference set. This begins, when the fitness exceeds a value of about 0.9.

12.2.4 *Deriving a 2D-Lookup matrix*

In the following, it is assumed, that the two operation images and the goal image are given. The question is how to derive a suitable 2D-Lookup matrix, which gives the best match between goal image and result image by the fitness measure given in the last subsection. The interesting point here is, that this derivation can be done be reusing this fitness measure. To prove this, assume a 2D-Lookup matrix, where all but one positions are set to White (1), and only a single position (g_1, g_2) is set to Black (0). Then, the 2D-Lookup will give a resultant image with all positions (x, y) set to Black, for which operation 1 yielded pixel value g_1 and operation 2 yielded pixel value g_2. Usually, there will be only a few black pixels within the result image. Now, as it was remarked in the last subsection, the fitness measure will give values above 0.9, if the set of black pixels lies completely within the reference, no matter, how many pixels are there. So, a criterion can be given for setting a pixel to Black or White in the 2D-Lookup matrix.

Let $l_{(x,y)}$ be a 2D-Lookup matrix constituted by setting only the pixel at (x, y) to Black, and $r_{(x,y)}$ be the result of the 2D-Lookup with the operation images 1 and 2 and this 2D-Lookup matrix. Then

$$l(x,y) = \begin{cases} \text{Black}, & \text{if } f(r_{(x,y)}) > 0.88 \\ \text{White} & \text{otherwise.} \end{cases}$$

In case there are no black pixels in $r_{(x,y)}$ at all, $l(x, y)$ is set to Gray, which stands for positions within the 2D-Lookup matrix, whose pixel value pairs do never occur within the operation images 1 and 2 at the same location. The value 0.88 has been chosen instead of 0.9 to give the adaptation some tolerance.

Figure 12.6 shows the result of this derivation for two example images. This procedure, which resembles a relaxation procedure, gives a quasi-optimal 2D-Lookup matrix for given operation images 1 and 2. The following subsection describes the manner, by which the two operations needed are derived by an individual of a GP.

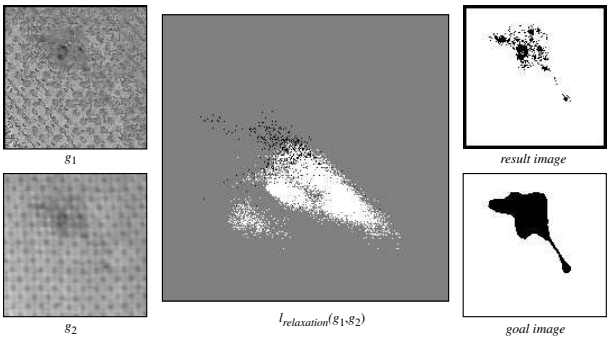

Fig. 12.6 The relaxation procedure for deriving the optimal 2D-Lookup matrix $l_{relaxation}(g_1, g_2)$, once the operation images g_1 and g_2 are given.

12.2.5 *Building up a GP individual*

In the following the structure of an individual of the GP population is described. On its base, there are formal superoperators, acting on an input image $o(x, y)$ and giving the result image $g(x, y)$. These superoperators make use of a common parameter structure PS, which contains the following entries:

- A *mask* $\mathcal{M} = \{(i, j)\}$ as a set of offset positions. The element k of $\mathcal{M}$ is assigned by $\mathcal{M}_k = (i_k, j_k)$. Applying a mask onto a pixel with position (x, y) gives a set of pixel positions $\mathcal{M} \circ (x, y) = \{(u, v) \,|\, u = x + i, v = y + j, (i, j) \in \mathcal{M}\}$, the "neighborhood" of (x, y). The cardinality of $\mathcal{M}$ is $|\mathcal{M}| = m$. Masks used in a PS are restricted to 3×3 symmetric masks.
- A *weighted mask* $\mathcal{M}_w$ is a tupel $(\mathcal{M}, f)$ of a mask $\mathcal{M}$ and a function $f : \mathcal{M} \to \mathcal{R}$, which assigns a weight value to each mask offset (i, j) of the mask $\mathcal{M}$. For convenience, mask and its weightings are pictured by a scheme as for a 3×3 mask:

$$\mathcal{M}_w = \begin{array}{|c|c|c|} \hline w_{(-1,-1)} & w_{(0,-1)} & w_{(1,-1)} \\ \hline w_{(-1,0)} & w_{(0,0)} & w_{(1,0)} \\ \hline w_{(-1,1)} & w_{(0,1)} & w_{(1,1)} \\ \hline \end{array}$$

  Weight values are restricted to the range $[-5, 5]$. If only $\mathcal{M}_w$ is given, $\mathcal{M}$ can be found as set of all offset positions (i, j) with $f_w(i, j) \neq 0$.

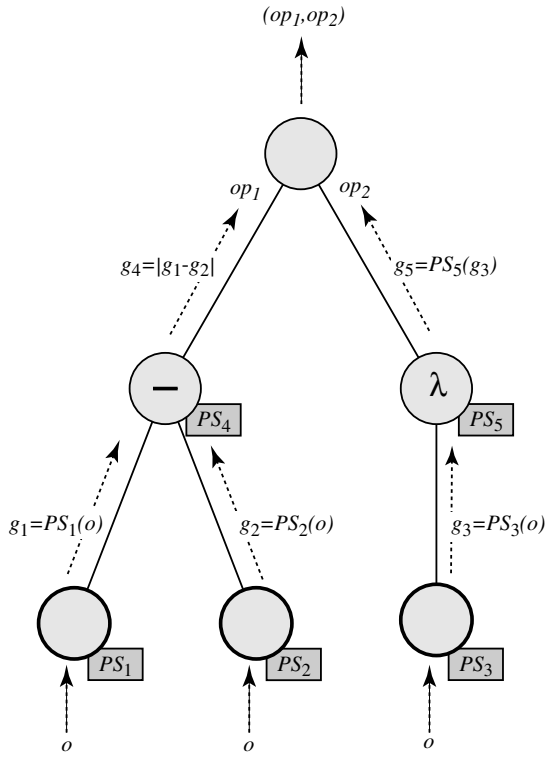

Fig. 12.7 Example for an individual GP tree, representing two image processing operations (for details, see text).

- A single offset vector $\vec{p} = (\delta_x, \delta_y)$. The magnitude of the components of the offset vector does not exceed 2.
- A BOOLEAN operation f_{bit} from $\{0,1\} \times \{0,1\}$ onto $\{0,1\}$ with number $N_{f_{bit}}$[‡]. The notation $f_{bit}(n,m)$ means applying f_{bit} bitwise, i.e. if $bit_i(n)$ extracts bit i of the number n in dual representation, then $bit_i(f_{bit}(n,m)) = f_{bit}(bit_i(n), bit_i(m))$.
- A permutation Π of the first m integers with Π_k the position of number k in the permutation.
- A ranking vector $\vec{v} = (v_1, v_2, \ldots, v_m)$, which assigns weights to a set of m sorted values. In the following, $rank_i^{max}(\mathcal{A})$ is the i-th

[‡]There are 16 such operations, and N refers to a given ordering of them, e.g. by lexicographical ordering of the two bit operands.

largest value of the value set $\mathcal{A}$, $rank_i^{min}(x,y)$ is the i-th smallest value. The components of $\vec{v}$ are restricted to the set $\{-1,0,1\}$.

- A symbolic mode value *mode*, which specifies which kind of superoperator is applied, using some of the parameters defined above. The mode symbols and their meaning are explained below.

Hence, a PS is a 6-tupel $(\mathcal{M}_w, \vec{p}, N_{f_{bit}}, \Pi, \vec{v}, mode)$.

In the following, all operations are to be restricted within the given image boundaries. Then, the following formal superoperators are introduced:

(1) Logical translation **TRANS**. It uses the vector $\vec{p}$ and the bit operation number $N_{f_{bit}}$. The operated image at position (x,y) is given by

$$g(x,y) = f_{bit}\left(o(x-\delta_x, y-\delta_y), o(x,y)\right).$$

(2) Convolution **CONVOL**. It uses the weighted mask $\mathcal{M}_w$. The convolved image is given by

$$g(x,y) = \sum_{(i,j)\in\mathcal{M}} f(i,j)\, o(x+i, y+j)$$

The weighted mask is zero-biased before operating with it, i.e., a value δ is chosen so that $\sum(f_w(\mathcal{M}_k)+\delta) = 0$.

(3) Ordered Weighted Averaging **OWA** [**?**]. This operation is used in fuzzy inference systems for defuzzification. It uses the ranking vector $\vec{v}$. Applying it as an image processing operation goes on as follows: the pixel values of the image function o of the original image in the neighborhood $\mathcal{M} \circ (x,y)$ are sorted in descending order, and the sorted value at position k is multiplied with the weight v_k. All prodcuts are summed up.

$$g(x,y) = \sum_{k=1}^{|\mathcal{M}|} v_k\, rank_k^{max}(o \circ (\mathcal{M} \circ (x,y)))$$

Some OWA weights represent well-known image processing operations. For example $(1,0,\ldots,0)$ as the dilation, $(0,\ldots,0,1)$ as the erosion, $(1,\ldots,-1)$ as the morphological gradient, (0, 0, ..., 0, 1, 0, ..., 0) as a ranking operator and $(1/n, 1/n, \ldots, 1/n)$ as the averaging operation. OWA allows for formalizing these operations within a single expression.

(4) Texture Numbers **TN**. Here, the per Π is used. At position (x, y), the operation derives a bit vector b with elements from $\{0, 1\}$ of the same size as $\mathcal{M}$, from the input image o. The element b_k of b is obtained by

$$b_k = \begin{cases} 1 & \text{if} \quad (o \circ (\mathcal{M} \circ (x, y)))_k \geq o(x, y), \\ 0 & otherwise. \end{cases}$$

Now, a number n is constructed from b as follows: bit k of n is given by the element l of the sequence b, with l being the index of k in the permutation Π: $bit_k(n) = b_{\Pi_k}$. Then, $g(x, y) = n$. The resultant values n are automatically rescaled to the grayvalue range $\{0, \ldots, g_{max}\}$.

(5) Ordered Weighted Minimum **OWM** [**?**]. This operation uses the weighted mask $\mathcal{M}_w$. The mask weights are sorted in descending order, the pixel values in the neighborhood of (x, y) are sorted in ascending order. Then, the minmax of these two value sequences is computed as value of the result image at position (x, y):

$$g(x, y) = \min_{k=1}^{|\mathcal{M}|} \left[\max \left[rank_k^{max}(f_w \circ \mathcal{M}), rank_k^{min}(o \circ (\mathcal{M} \circ (x, y)) \right] \right].$$

From this, an image operation is fully specified by means of a PS. The GP individual trees are constructed according to the following rules:

(1) At the root level, every tree has two branches (in order to have two operations for the 2D-Lookup).
(2) At each level, each function node branches into a set of function nodes and terminals of the next lower level.
(3) To each function node and terminal, a randomly initialized PS is assigned.

To each function node, an image operation out of the following set is assigned (here, g_1, g_2 are the operands, g is the result of the operation):

(1) Pixelwise Subtraction $-$. $g(x, y) = |g_1(x, y) - g_2(x, y)|$.
(2) Pixelwise Squaring **sq**. $g(x, y) = g_1^2(x, y)/g_{max}$.
(3) Pixelwise Minimum **min**. $g(x, y) = \min\{g_1(x, y), g_2(x, y)\}$.
(4) Pixelwise Maximum **max**. $g(x, y) = \max\{g_1(x, y), g_2(x, y)\}$.
(5) Performing PS λ. $g = PS \circ g_1$.

A tree T generates the two operation images needed for 2D-Lookup from the original image in the following bottom-up manner. The original image o is given to all terminals. Each terminal applies its PS onto o. Then, the processed images are given as operands to the nodes at the next upper level, processed, given to the next upper level and so forth. The root node collects its both operand images, performs the relaxation procedure, which was described in the last subsection, and computes the fitness measure f for the now fully specified 2D-Lookup algorithm. In this manner, a fitness f is assigned to a tree T.

As an example, consider the six-node GP individual in Fig. 12.7. Five random initialized $PS = \{\mathcal{M}_w, \vec{p}, N_{f_{bit}}, \Pi, \vec{v}, mode)$ are given:

$$PS_1 = \left\{ \begin{array}{|c|c|c|} \hline 0 & 2 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 3 & 0 \\ \hline \end{array}, (1,1), 11, \Pi_{213}, (1,0,0), \text{ TRANS} \right\}$$

$$PS_2 = \left\{ \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 1 & 2 & 2 \\ \hline 0 & 0 & 0 \\ \hline \end{array}, (1,2), 7, \Pi_{132}, (1,0,0), \text{ OWA} \right\}$$

$$PS_3 = \left\{ \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & 2 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}, (-1,0), 3, \Pi_{12354}, (0,0,1), \text{ OWA} \right\}$$

$$PS_4 = \left\{ \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 3 & 5 & 2 \\ \hline 0 & 0 & 0 \\ \hline \end{array}, (-1,-2), 5, \Pi_{312}, (1,0,-1), \text{ TN} \right\}$$

$$PS_5 = \left\{ \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & 2 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}, (2,0), 5, \Pi_{51243}, (1,0,0), \text{ OWA} \right\}$$

Note, that PS_4 is never applied onto an image, and PS_3 and PS_5 share the same mask. Bit operation 7 is assumed to be the XOR operation. From this, operation 1 of the tree is specified as absolute difference of the original image translated and XOR-ed by the offset (1,1) (PS_1), and the original image dilated by a horizontal line-mask (PS_2). Operation 2 is given

as morphological opening of the original image by a cross-mask (PS_3 and PS_5). Fig. **??** shows the intermediate and operation images of this tree structure.

The structuring of image processing operations by the trees has been chosen in this rather complicated manner for the following reasons:

- The operations, which are represented by such trees, resemble well-known image processing operations. A tree is likely to represent operations as dilation, erosion, closing, opening, morphological gradients, SOBEL operator, statistical operators, GAUSSIAN filtering, shadow images and so forth.
- The represented operations are unlikely to give unwanted operation images, which are completely white or black.
- They preserve image locations.
- The maximum arity of a node is two. Also, maximum tree depth was restricted to five. This was set in order to allow for the maintenance of the obtained trees, e.g. for manually improving the designed filters by removing redundant branches. Processing time is kept low, too (but processing time does not go into the fitness function itself!).

Finally, figure **??** gives some operation images obtained from the same original image by different randomly constructed and configured trees. These images demonstrate the variabilty of the generated operations, each of which enhances or surpresses different image substructures, and none of which gives a trivial image operation.

12.3 Results

To learn about the framework's abilities, textile images were used which were taken from the "Textilfehler-Katalog" of the IPK Berlin (TFK). Four examples will be given here. At first, a conventional feature classification approach was applied. Co-occurrence features of 10 8×8 texture windows for each class (background texture and fault) were computed. Then, for each type of texture a multilayer backpropagation neural network (MBPN) with 14 input neurons (for the 14 co-occurrence features), 16 hidden neurons and 2 output neurons (for the two classes) for each sample was trained over 1000 cycles. The trained MBPNs were recalled on the original images. The

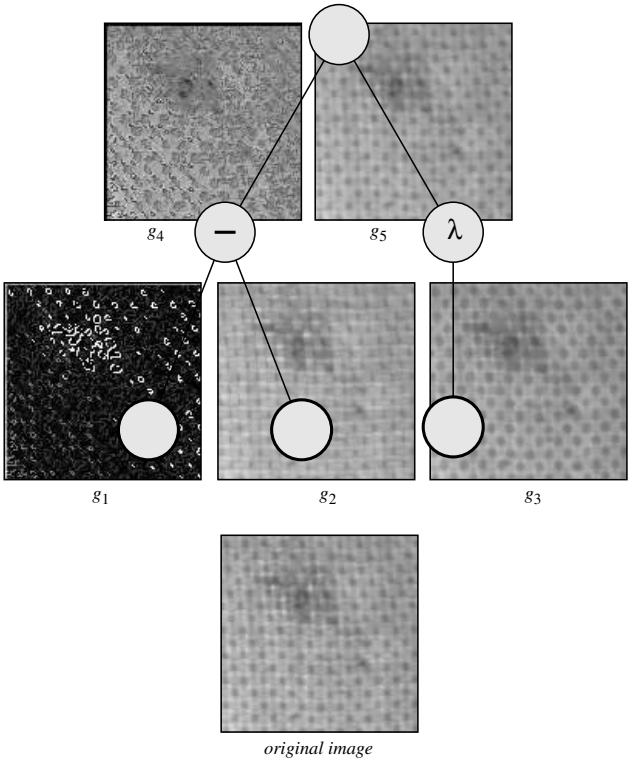

Fig. 12.8 Intermediate images, as generated by the tree in Fig. 12.7.

results are shown in Fig. **??**. In general, the resolution of the method is not very high, since it is restricted to the texture window size of 8×8. The results for TFK 1-fr8 and TFK 1-st11 are good. The result for TFK 3-st23b contains many errors. This is due to the varying grayvalue appearance of the fault region. The MBPN can not generalize good in this case. The result for TFK 3-fr1a is inacceptable. The fault is very small and of low contrast. Too many conflicting grayvalue constellations appear within the image. They can't be classified by a MBPN.

Of course, these results will not prove, that the feature classification approach in general is not able to perform better. The approach may be adapted for each sample. But it has to be noted, that there is no other way for improving the recall results in this case than by either supplying more training samples, training the MBPN longer or changing the feature

Results

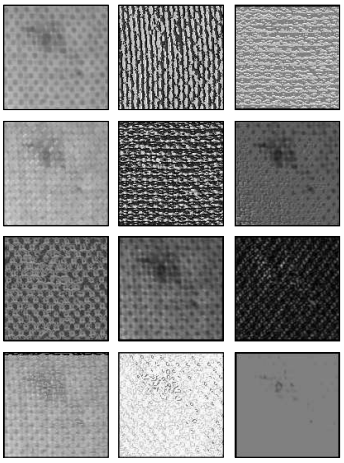

Fig. 12.9 Some operation images, as generated from the upper left image by randomly generated trees.

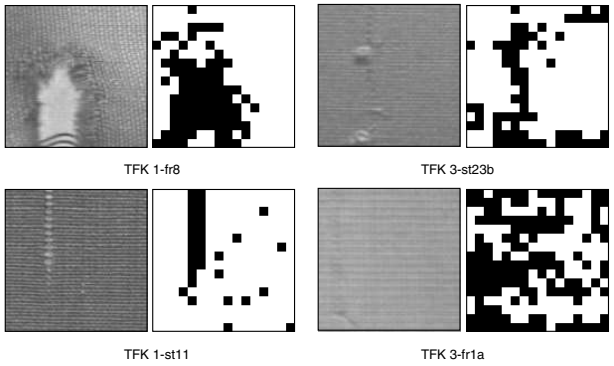

Fig. 12.10 Results for applying the conventional texture classification approach to four examples of the TFK (details see text).

calculation method.

For instructing the LUCIFER2 frameworking approach, subimages were cut from the whole images, containing the fault and its neighboring texture. Goal images were drawn by using a photo retouching program. Then, the framework is started for each of the four samples. The results, including both operation images and the relaxed 2D-Lookup matrix, are given in the figures **??** to **??**. For each problem, a population of 30 individuals, with 80

children in each generation, were used. The run went on until the genetic diversity of the population vanished (all individuals have the same fitness). Usually, this was the case after performing about ten generations.

As can be seen from the figures, the framework fulfills its task and designs appropriate texture filters. The performance is usually good and sometimes very good. It is to be noted, that *exact* resemblance will not be possible, and it would not be of any merit. The visual appearance of the fault border is a subjective one, and the user might give a slightly modified goal image as well. The goal image contains *virtual* borders.

Also, the framework will never find the (globally) optimal texture filter, since the searchspace is much too large to be covered by the genetic search. The most important aspect here is that there are many good solutions for a given task. The framework randomly evolves the initialized solutions towards better ones.

The generalization ability of the designed texture filters was checked on a larger test suite, also taken from the TFK. For 24 texture fault problems, small reference images were cut out and goal images were drawn. For each fault, the framework was run exactly once. The evolved filter was applied to the whole image. The number of black pixels out of the reference image part was counted, as well as the number of gray pixels. The first value gives a measure for the error rate ("false alarms") of the texture filter, the second a measure for the compatibility of reference image and total image. The results are given in Table **??**. The average error rate is 1.9% and the average incompatibility is 2.2%. These are good results, too. However, the variations of the error rate are comparatively large (between 2 and 17278 pixels out of 262144 pixels, with one out-layer of 57774 pixels).

A consideration of the 2D-Lookup matrices helps to explain this fact. The following can be said:

- Some matrices seems to be separable, i.e. the textured background is represented by a group of compact white regions. The fault appearance (the black dots) surrounds these regions. In this case, the filter is expected to have a good generalization ability.
- There are regions, where black and white dots are hoplessly intermingled. The more such regions are found within a matrix, the more the evolutionary adaptation features random grayvalue constellations within the two operation images. In this case, the filter will perform bad on other images with the same texture and fault

categories.

Compactness within the 2D-Lookup matrices is considered as main provision for filter's higher generalization ability. If the matrices are manipulated in a manner, which enhances compactness of its black and white regions, the filter will perform better on newly presented images (possibly for the price of a slightly lower performance on the input image, from which the filter was designed). Future work will focus on this.

Table 12.1 Generalization ability of the LUCIFER2 framework for 24 texture fault examples taken from "Textilfehler-Katalog." All images contains 262144 pixels. On an average, there are 1.9% of the black pixels wrong, and 2.2% of the pixels gray.

TFK No.	No. of Wrong Blacks	No. of Grays
1-fr13a	407	12832
1-fr13b	11	33030
1-fr13c	2	2908
1-fr13d	18	1343
1-fr19	15	1322
1-st47a	1017	2276
2-i53	75	6
2-lau11b	70	2098
2-lau19	280	862
2-lau2	524	3804
2-lau6	498	495
2-lau24	614	283
3-fr4c	2325	16301
3-fr8	57774	5552
3-st18b	5717	10352
3-st20	2142	3367
3-fr19	15044	6216
3-st47a	17278	17680
4-fr1a	88	2004
4-st14	2355	3882
4-fr19	67	1854
4-st22a	536	2280
4-st27a	5	2555
4-st29	12135	6450

12.4 Conclusions

A framework was presented, which allows for the design of texture filters for fault detection (two class problem). The framework is based on the 2D-Lookup algorithm, where two filter output images are used as input.

The approach was applied to four texture problems and the performance of the framework was discussed. The results, obtained without "human intervention," are ready-to-use texture filters. Also, they can be tuned in order to obtain even more better results, or combined in a superposed inspection system. The following are our experiences during the test runs:

- The framework was able to design texture filters with good or very good performance.
- The goal image matched the fault region quite satisfactorily.
- Bordering regions should be neglected for fitness evaluation.
- The framework was able to design filters for the detection of non-compact fault regions and fault regions with varying appearance.
- The designed filters may be subjected to further improvements by the user.

Current work focuses on several improvements of the whole architecture, especially on the inclusion of rescaling and rescanning into the designed filter operations, and on an evaluation of the 2D-Lookup matrix by neural networks in order to get a comprehensive solution for a given texture filtering problem.

Acknowledgment

This research is supported by the Deutsche Forschungsgemeinschaft (DFG), Schwerpunktprogramm "Automatische Sichtprüfung technischer Objekte", EPISTO, Ni 473/1-2. The authors wish to thank Dr. B. Schneider to make the "Textilfehler-Katalog" accessible to this research (the "Textilfehler-Katalog" is accessible via *http://vision.fhg.de/ipk/tfk*) and to A. Zentner for implementing an on-line version of the LUCIFER2 framework (URL http://vision.fhg.de/ipk/demo/lucifer2).

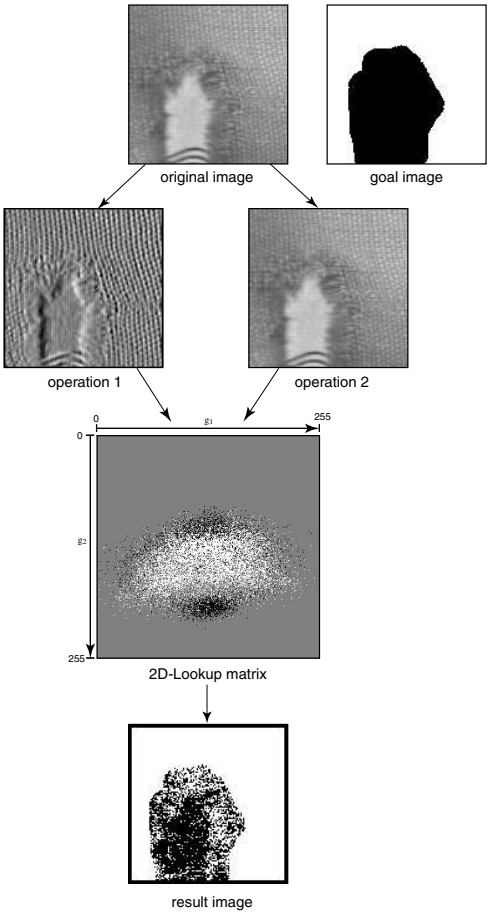

Fig. 12.11 Results for example 1 (TFK 1-fr8).

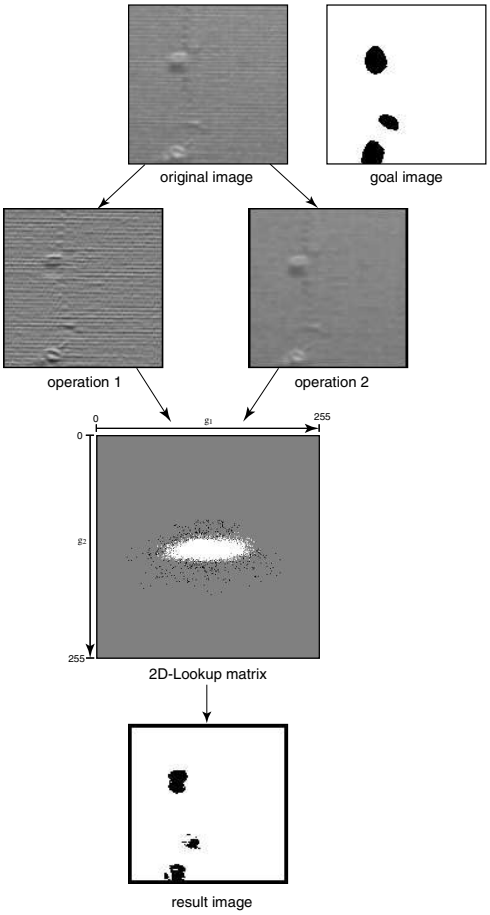

Fig. 12.12 Results for example 2 (TFK 3-st23b).

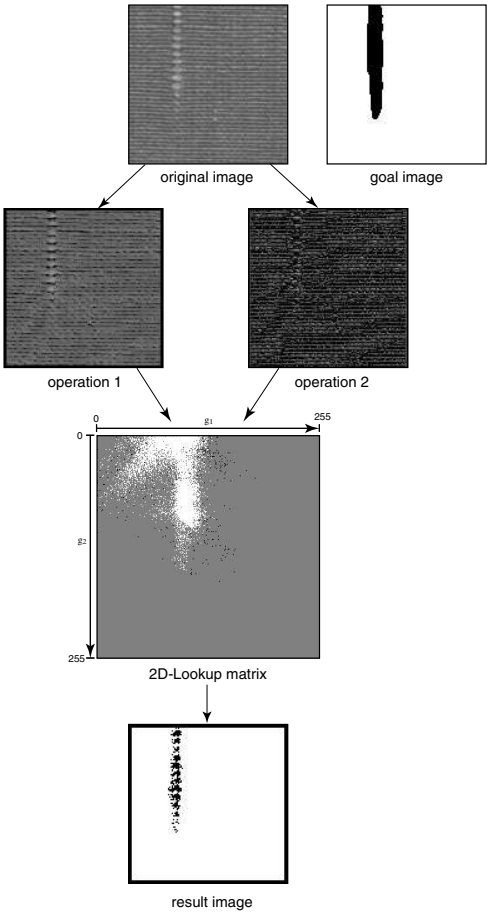

Fig. 12.13 Results for example 3 (TFK 1-st11).

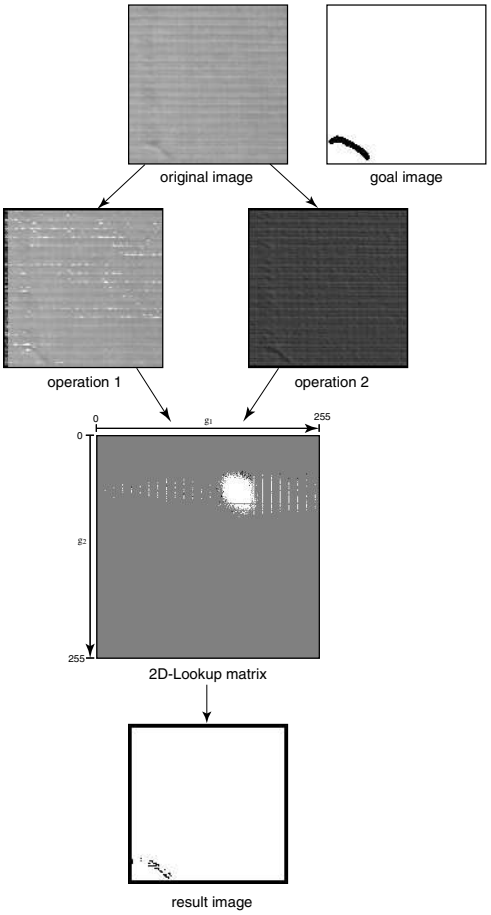

Fig. 12.14 Results for example 4 (TFK 3-fr1a).

References

[1] Dubois, D., Fargier, H., Prade, H., "Beyond min aggregation in multicriteria decision: (ordered) weighted min, discri-min, lexmin", in: Yager, R. R., Kacprzyk, J. (eds.), "The ordered weighted averaging operators — Theory and applications", Kluwer Academic Publishers, Dordrecht a.o., 1997.

[2] Goldberg, D. E., "Genetic algorithms in search, optimization & machine learning", Addison-Wesley, Reading, MA, 1989.

[3] Gonzales, R. C., Woods, R. E., "Digital Image Processing", Addison-Wesley, Reading MA, 1993.

[4] Haralick, R., Shanmugam, K., Dinstein, I., "Textural features for image calssification", IEEE Trans. SMC, 3, (6), pp.610-621, 1973.

[5] Haralick, R., Shapiro, L., "Image segmentation techniques", Computer Vision, Graphics and Image Processing, 29, pp.100-132, 1985.

[6] Holland, J. A., "Adaptation in natural and artificial systems", MIT Press, Cambridge MA, 1975.

[7] Köppen, M., Ruiz-del-Solar, J., Soille, P., "Texture segmentation by biologically-inspired use of neural networks and mathematical morphology", Proc. NC'98, Vienna, Austria, pp.267-272, 1998.

[8] Köppen, M., Ruiz-del-Solar, J., "Fuzzy-based texture retrieval", Proc. FUZZ-IEEE'97, Barcelona, Spain, pp.471-475, 1997.

[9] Köppen, M., Teunis, M., Nickolay, B., "A framework for the evolutionary generation of 2D-Lookup based texture filters", Proc. IIZUKA'98, Iizuka, Japan, pp.965-970, 1998.

[10] Köppen, M., Soille, P., "Two-dimensional frameworks for the application of soft computing to image processing", Proc. IWSCI'99, Muroran, Japan, pp.204-209, 1999.

[11] Koza, J., "Genetic programming — On the programming of computers by means of natural selection", MIT Press, Cambridge, MA, 1992.

[12] Langdon, W. B., Poli, R., "Fitness Causes Bloat", Proc. of the 2nd On-line World Conference on Soft Computing in Engineering Design and Manufacturing (WSC2), 1997.

[13] Serra, J., "Image analysis and mathematical morphology", Academic Press, London, 1982.

[14] Serra, J., "Image analysis and mathematical morphology. Vol. 2: Theoretical advances", Academic Press, London, 1988.

[15] Whitley, D., "A genetic algorithm tutorial", Statistics and Computing, 4, pp.65-85, 1994.

[16] Yager, R. R., "On ordered weighted averaging aggregation operators in multi-criteria decision making", IEEE Trans. SMC, 18, pp.183-190, 1988.