21

Non Linear Algorithm for Automatic Image Processing Applications in FPGAs

Emerson Carlos Pedrino[1], Valentin Obac Roda[2] and Jose Hiroki Saito[1,3]
[1]*Federal University of Sao Carlos, Department of Computer Science*
[2]*Federal University of Rio Grande do Norte, Department of Electrical Engineering*
[3]*Faculty of Campo Limpo Paulista*
Brazil

1. Introduction

Mathematical morphology supplies powerful tools for low-level image analysis. The design of morphological operators for a given application is not a trivial one. For some problems in low level image processing the best result is achieved applying to the image an ordered sequence of morphological operators, that can be done manually, but is not easy and not always leads to the best solution. Genetic programming (GP) is a branch of evolutionary computing, and it is consolidating as a promising method for applications of digital image processing. The main objective of genetic programming is to discover how computers can learn to solve problems without being programmed for that. In the search for a practical automatic solution for low level image processing using mathematical morphology and genetic programming we present in this chapter a Matlab algorithm used for this purpose. Two sample images feed the Matlab application, the first one the original image with all defects, the second one the goal image where the defects of the original image were corrected. If we want to find the mathematical morphology operators that implement a certain filter that removes specific noise from the image, we supply a noisy image and an image were the noise was removed. The second image can be obtained from the noisy image applying an image manipulation program. After the parameters are supplied to the Matlab algorithm, the developed program starts to search for the sequence of morphological operators that leads to the best solution. The program works iteratively, and at each iteration compares the result of the morphological operations applied to the image set with the previous ones. To quantify how good is the solution at each iteration the resulting image is compared with the reference image using the mean absolute error (MAE) of the pixels. The best solution of the process is the image from a certain set whose error is less than a reference error indicated to the function. Using this methodology it was possible to solve a number of low level image processing problems, including edge detection, noise removal, separation of text from figures, with an error less than 0.5%, most of the time. Examples are presented along the text to clarify the use of the proposed algorithm. In addition, the sequence of operators obtained by the Matlab procedure was used to reconfigure an hardware architecture implemented in FPGAs to process images with the generated instructions in real time.

2. Theoretical background

In this section it will be presented a brief review of the theoretical background needed to understand the concepts used in the development of the current work. Firstly, it will be presented the theoretical basis of mathematical morphology followed by the fundamentals ideas of the genetic programming approach.

2.1 Mathematical morphology

Morphological image processing is a nonlinear branch in image processing developed by Matheron and Serra in the 1960´s, based on geometry, and on the mathematical theory of order (Dougherty, 1992; Serra, 1982; Weeks, 1996; Soille, 1999; Sonka et al., 1993; Facon, 1996). Morphological image processing has proved to be a powerful tool for binary and grayscale image computer vision processing tasks, such as edge detection, noise suppression, skeletonization, segmentation, pattern recognition, and enhancement. Initial applications of morphological processing were biomedical and geological image analysis problems. In the 1980´s, extensions of classical mathematical morphology and connections to other fields were developed by several research groups worldwide along various directions, including: computer vision problems, multi scale image processing, statistical analysis, and optimal design of morphological filters, to name just a few (Pedrino et al., 2010). The basic operations in mathematical morphology are the dilation and the erosion, and these operations can be described by logical and arithmetic operators. Dilation and erosion morphological operators can be represented respectively by the sum and subtraction of Minkowski sets (Dougherty, 1992):

$$A \oplus B = \cup\{B+a \mid a \in A\} \tag{1}$$

$$A \ominus -B = \cap\{A+b \mid b \in B\} \tag{2}$$

In Equation (1), A is the original binary image, B is the structuring element of the morphological operation, and B+a is the B displacement by a. Therefore, the dilation operation is obtained by the union of all B displacements in relation to the valid A elements. In Equation (2), -B is the 180^{o} rotation of B in relation to its origin. Therefore, the erosion operation corresponds to intersection of the A displacements by the valid points of -B. According to Equation (1), the dilation operation will expand an image, and the erosion operation will shrink it. These operations are fundamental to morphological processing, and many of the existing morphological algorithms are based on these two primitives operations. These ideas can be extended to gray level image processing using maximum and minimum operators, too (Gonzalez & Woods, 2008). Many applications examples can be found in that text. In addition, color is known to play a significant role in human visual perception. The application of mathematical morphology to color images is difficult due to the vector nature of the color data. Mathematical Morphology is based on the application of lattice theory to spatial structures (Angulo & Serra, 2005). The definition of morphological operators needs a totally ordered complete lattice structure. A lattice is a partially ordered set in which any two elements have at least an upper bound (supremum) and a greatest lower bound (infimum). The supremum and the infimum are represented by the symbols $\vee$ and $\wedge$, respectively. Thus, a lattice is complete if every subset of the lattice has a single supremum and a single infimum. The application of mathematical morphology to color images is difficult due to the vector nature of the color data. The extension of concepts from grayscale morphology to color morphology must first choose an appropriate color ordering,

a color space that determines the way in which colors are represented, and an infimum, and a supremum operator in the selected color space should also be defined. There are several techniques for ordering vectors. The two main approaches are marginal ordering and vector ordering. In the marginal ordering, each component P of a pixel is ordered independently, and the operations are applied to each channel; unfortunately, this procedure has some drawbacks, e.g., producing new colors that are not contained in the original image and may be unacceptable in applications that use color for object recognition. The vector ordering method for morphological processing is more advisable. Only one processing over the three dimensional data is performed using this method. There are several ways of establishing the order, e.g., ordering by one component, canonical ordering, ordering by distance and lexicographical ordering (Chanussot & Lambert, 1998). Once these orders are defined, then the morphological operators are defined in the classical way (Pedrino, 2010).

2.2 Genetic programming

Genetic programming (GP) is a technique for automatic programming nowadays and may provide a better context for the automatic generation of morphological procedures. GP is a branch of evolutionary computation and artificial intelligence, based on concepts of genetics and Darwin's principle of natural selection to genetically breed and evolve computer programs to solve problems (Koza, 1992). Genetic Programming is the extension of the genetic algorithms (Holland, 1975) into the space of programs. That is, the objects that constitute the population are not fixed-length character strings that encode possible solutions to a certain problem. They are programs (expressed as parse trees) that are the candidate solutions to the problem. There are few applications of GP for the automatic construction of morphological operators. According to Koza, in GP, populations of many computer programs are genetically bred by means of the Darwinian principle of survival and reproduction of the fittest individual in a population. In this approach, GP starts with an initial population of computer programs generated randomly, in which each program is represented by functions and terminals (operands) appropriate to a certain problem domain. Each chromosome (computer program) in the population is measured in terms of its fitness measure. This measure indicates how well a particular individual performs in a particular problem environment. The nature of the fitness depends on the problem at hand. A new offspring population of chromosomes is generated based on the current population using the Darwinian principle of reproduction and survival of the fittest, as seen before. In addition, the genetic crossover operator is used, too. The reproduction operator can copy, in proportion to a fitness, a chromosome (computer program) from current population into the new population. The crossover operator can produce new offspring computer programs from two parental chromosomes based on their fitness. Typically, the programs are of different sizes and shapes. After these process, a new population of individuals is generated and the old one is deleted. This process is repeated for many generations until a desired result can be obtained.

3. Developed system

The developed system for automatic construction of morphological operators uses a genetic programming algorithm that operates with two images from a certain image set, an input image, and an image containing only features of interest which should be extracted from the input image. The genetic procedure looks in the space of mathematical morphology operators for sequences that allow extracting the features of interest from the original image. The operators are predefined procedures from a database that work with any kind of

structuring elements having different shapes and sizes. It is also possible to include new operators in the database when necessary. The program output is a tree based structure containing the best individual of the final population. The genetic algorithm parameters are supplied by the user using a text user interface. The main parameters are: tree depth, number of chromosomes, number of generations, crossover rate, mutation rate, and certain kinds of operators suited to a particular problem. The mean absolute error (MAE) was used as a fitness measure. The cost function using the MAE was calculated as follows:

$$d(a,b)=\sum_i\sum_j |(a(i,j)-b(i,j)|/XY \quad (3)$$

In Equation (3), 'a' is the resulting image evaluated by a particular chromosome (program), and 'b' is the goal image with the same size as 'a', and '(i,j)' is the pixel coordinate. The programs are encoded as tree structure chromosome. The main steps of the proposed algorithm are illustrated in Figure 1. In the Figure, the index 'i' refers to an individual in the population. The reprodution rate is 'pr', the crossover rate is 'pc', and the mutation rate is 'pm'. The goal image can be created using an editor program. As Figure 1 shows, initially the genetic parameters are selected by the user, along with a couple of sample images that represent the problem to be solved. Then, the genetic procedure generates a random population of computer programs (chromosomes), according to the user specifications. A fitness value is assigned to each program, after the operations of reproduction, crossover and mutation, a new population of individuals is generated. The described evolutionary process is repeated by many generations and can be stopped according to a stop condition. The mutation operator, that was not previously discussed, simply generates an individual belonging to the space of solutions of the problem and connects it to a random point of a particular randomly chosen chromosome. Such operation is performed with a given probability.

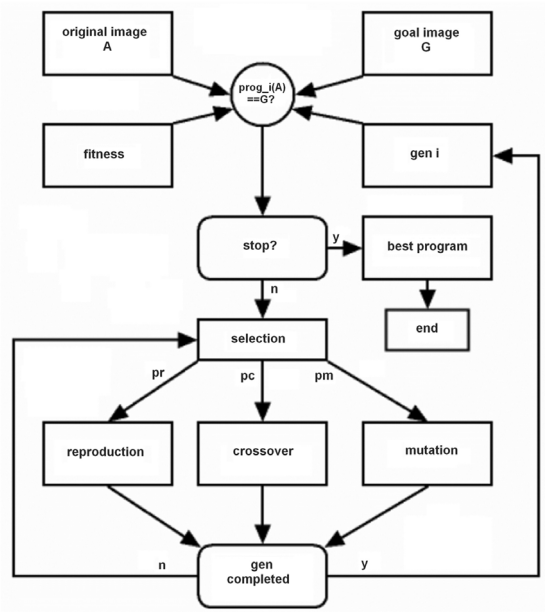

Fig. 1. Flowchart of developed system

The process of evaluating a given depth four chromosome of the population of individuals is shown in Figure 2. 'Img_in' corresponds to the input image, 'ero' and 'dil' instructions corresponds to the erosion and dilation operations respectively. The argument 'end_n' matches an address of a table containing all combinations of structuring element for a given problem. Both the instructions, and the arguments are found in an intelligent manner when a pair of input images are presented to the genetic procedure. It can be seen in Figure 2 that initially the input image is eroded, followed by two dilations and the resulting image is eroded again. All the morphological operators in the example use the same structuring element pointed by 'end_n'.

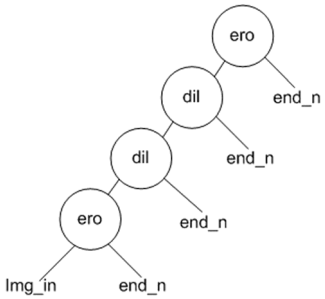

Fig. 2. Example of a chromosome representing a morphological filter and its arguments

The tournament selection method was the one chosen to be used in this work (Koza, 1992). The training set, used to extract the pair of images used in the presented method, was obtained using samples of synthetic images of various resolutions. For each resolution the maximum depth size for each chromosome tree and the error calculation functions were changed. The training will be further detailed in Section 3.1. As follows, the Matlab developed algorithm is presented along with some examples. In all the examples binary images were used, however, the method can be extended to handle any type of image.

3.1 Matlab algorithm

The Matlab algorithm for binary images mathematical morphology automatic processing developed in this work is presented as follows.

```
% - Developed Algorithm
% - pop: Initial Population
% - nc: number of chromosomes
% - cr: randomly generated chromosome
% - cr_col: columns of cr
% - num_instr: number of instructions
% - profd: maximum tree depth
% - arg: pointer to the table of arguments
pop=pop_init(nc,cr,cr_col,num_instr,profd,arg);
% - ger: generations
% - ng: number of generations
% - ct_aux: auxiliary cost
ger=0;
ct_aux=inf;
while (ger<=ng)
```

```
            % - ct_gn: costs from generation 'n'
            % - img_org: input image
            % - img_obj: goal image
            % - instr: instruction table
            ct_gn=cost(nc,pop,img_org,img_obj,instr,profd);
            % - elt: best program of generation 'n'
            % - ct_min: cost of best program
            [elt,ct_min]=elit_indv(pop,ct_gn);
            % - Elitism
            % - sol: best program found
            if (ct_min < ct_aux)
              ct_aux=ct_min;
              sol=elt;
            end
            % err: tolerated error
            if (ct_aux<=err)
              break;
            end
            % - Crossover
            % - pop_g: new population
            % - tx_cs: crossover rate
            tx_cs_conv=(-10*tx_cs/100)+10;

            pop_g=crossover(tx_cs_conv,pop,nc,img_org,img_obj,instr
      ,profd,sol);
            % - Mutation
            tx_mt_conv=(-10*tx_mt/100)+10;
            pop=mutation(pop_g,num_instr,profd,nc,tx_mt_conv,arg);
            ger=ger+1;
      end
```

According to the previous code, the function *pop_init* is responsible for generating randomly, using user provided parameters, the initial population of individuals. The parameters are described as follows.

- nc: number of chromosomes (programs);
- cr: initial chromosome user created;
- cr_col: number of columns of cr;
- num_instr: instruction number;
- profd: maximum depth tree tolerated;
- arg: pointers vector for the arguments table;

The implementation of the algorithm *pop_init* function is shown as follows. This function uses another function called *ger_cr* to generate chromosomes randomly.

```
      function p_i=pop_init(nc,cr,cr_col,num_instr,profd,arg)
      % Initial Population of Individuals
      pop(1,1:cr_col)=cr;
      for i=2:nc
              cr=gera_cr(num_instr,profd,arg);
              pop(i,:)=cr;
      end
      p_i=pop;
```

```
function cr=ger_cr(num_instr,profd,arg)
cr=zeros(1,profd*4);
[~,num_arg]=size(arg);
for i=1:4:profd*4
    aleat=round(rand*(num_instr-1));
    cr(i)=aleat;
    aleat=round(rand*(num_arg-1))+1;
    cr(i+1)=aleat;
    aleat=round(rand*(num_arg-1))+1;
    cr(i+2)=aleat;
    aleat=round(rand*(num_arg-1))+1;
    cr(i+3)=aleat;
end
```

The cost function is responsible for calculating the cost of each chromosome in current population of individuals. Internally, it uses two other functions, *build_op* and *comp*, the first one is responsible for building the program, according to the tree structure shown previously, and the second is responsible for calculating the fitness of chromosomes. Below, the code part related to the *cost* and *build_op* functions is presented. Following, the function *comp* is presented, too. The main parameters used for each function are the following.

- nc: number of chromosomes;
- pop: current population;
- img_org: input image;
- img_obj: goal image;
- instr: instructions vector;
- profd: maximum depth allowed to each chromosome.

The *build_op* function is responsible for the construction of each individual, using functions and arguments provided by the user through the instructions vector and by the table of arguments. The input image is applied to each program generated automatically and the *comp* function tests whether the object pixels and the image background pixels correspond to the pixels of the object and background of the goal image, thereby creating a vector containing all the costs associated to each program obtained in the present generation.

```
function c=cost(nc,pop,img_org,img_obj,instr,profd)
% - It Evaluates the Cost of Each Individual from the Current
Population
c=ones(1,nc);
for i=1:nc
    sample=build_op(instr,pop(i,:),profd,img_org);
    c(i)=comp(amostra_c,img_obj);
end


function op = build_op(instr,cr,profd,img)
% - It Builds a Program
%   to be Applied to the
%   Input Image
op=eval([char(instr(cr(1)+1)),'(','img',',',num2str(cr(2)),',
',num2str(cr(3)),',',num2str(cr(4)),')']);
```

```
aux_op=op;
% - Building
for i=5:4:profd*4
      op=eval([char(instr(cr(i)+1)),'(','aux_op',',',num2str(
cr(i+1)),',' ,num2str(cr(i+2)),',',num2str(cr(i+3)),')']);
      aux_op=op;
end


function comp_ab = comp(a,b)
% - It Compares two images (a and b)
TP=0.005;
FN=0.005;
TN=0.005;
FP=0.005;
[lin,col]=size(a);
for i=1:lin
    for j=1:col
       if b(i,j) && a(i,j)
           TP=TP+1;
       else
           if b(i,j) && ~a(i,j)
               FP=FP+1;
           else if ~b(i,j) && ~a(i,j)
                   TN=TN+1;
               else
                   if ~b(i,j) && a(i,j)
                       FN=FN+1;
                   end
               end
           end
       end
    end
end
SV=TP/(TP+FN);
SP=TN/(TN+FP);
comp_ab=1-(sqrt((1-SP)^2+(1-SV)^2))/sqrt(2);
comp_ab=1-comp_ab;
```

The *elit_indv* function returns the best current generation chromosome, the elite, and their associated cost; its parameters are the following.

- pop: current population;
- ct_gn: vector cost for the current population;

The implementation of the *elit_indv* function is shown as follows.

```
function [elt,ct_min]=elite_indv(pop,ct)
% Best Individual
[ct_min,ind]= min(ct);
elt=pop(ind,:);
```

The next function implemented was the genetic crossing (crossover), which can be seen below. Its main parameters are:

- tx_cs: crossing rate, between 0 and 1 (0 - 100%);
- pop: current population;
- nc: number of chromosomes;
- img_org: input image;
- img_obj: goal image;
- instr: instructions vector;
- profd: maximum depth allowed for each chromosome;
- sol: best chromosome (program) found so far.

The crossing is performed between two trees in the population of individuals, selected according to a given probability, user specified. The crossing method used is similar to the one implemented by Koza (Koza, 1992).

```
function
pop_g=crossover(tx_cs,pop,nc,img_org,img_obj,instr,profd,sol)
% - Crossover.
pop_g=pop;
pop_g(1,:)=sol;
pop_indv_at=2;
while (pop_indv_at<=nc)
    rd_c=round(rand*nc+.5);
    p1=pop(rd_c,:);
    rd_c=round(rand*nc+.5);
    p2=pop(rd_c,:);
    rd_c=round(rand*nc+.5);
    p3=pop(rd_c,:);
    px=[p1;p2;p3];
    % - Tournament selection.
    ct_gn=custo(3,px,img_org,img_obj,instr,profd);
    [elt,~]=elite_indv(px,ct_gn);
    % father
    p=elt;
    [~,ind_p2]=sort(ct_gn);
    % mother
    m=px(ind_p2(2),:);
    rd_num=rand*10;
    if (rd_num>tx_cs)
        ind_imp=1:4:profd*4;
        [~,c_ind_imp]=size(ind_imp);
        ind_p=ind_imp(round(rand*(c_ind_imp)+0.5));
        ind_m=ind_imp(round(rand*(c_ind_imp)+0.5));
        gen_p=p(ind_p);
        gen_m=m(ind_m);
        p(ind_p)=gen_m;
        m(ind_m)=gen_p;
        aux=round(rand*(3)+0.5);
        crt_p=ind_p+aux;
        crt_m=ind_m+aux;
        gen_p=p(crt_p:ind_p+3);
        gen_m=m(crt_m:ind_m+3);
        p(crt_p:ind_p+3)=gen_m;
```

```
            m(crt_m:ind_m+3)=gen_p;
            pop_g(pop_indv_at,:)=p;
            pop_indv_at=pop_indv_at+1;
            if (~rem(nc,2))
                pop_g(pop_indv_at,:)=m;
                pop_indv_at=pop_indv_at+1;
            end
        else
            pop_g(pop_indv_at,:)=p;
            pop_indv_at=pop_indv_at+1;
            if (~rem(nc,2))
                pop_g(pop_indv_at,:)=m;
                pop_indv_at=pop_indv_at+1;
            end
        end
    end
```

- The mutation function swaps parts of the selected programs, according to a given probability, with parts of programs belonging to the space of solutions of a given problem; its parameters are shown as follows.
- pop: current population;
- num_instr: number of instructions;
- profd: maximum depth allowed for each chromosome;
- nc: number of chromosomes;
- tx_mt: mutation rate, between 0 and 1 (0 - 100%);
- arg: pointers vector for the argument table.

```
function pop_g=mutation(pop,num_instr,profd,nc,tx_mt,arg)
% Mutation.
[~,num_arg]=size(arg);
pop_g=pop;
for i=1:nc
    for j=1:4:profd*4
        rd_num=rand*10;
        if (rd_num>tx_mt)
            rd=round(rand*(num_instr-1));
            pop_g(i,j)=rd;
        end
        rd_num=rand*10;
        if (rd_num>tx_mt)
            rd=round(rand*(num_arg-1))+1;
            pop_g(i,j+1)=rd;
        end
        rd_num=rand*10;
        if (rd_num>tx_mt)
            rd=round(rand*(num_arg-1))+1;
            pop_g(i,j+2)=rd;
        end
        rd_num=rand*10;
        if (rd_num>tx_mt)
            rd=round(rand*(num_arg-1))+1;
```

```
                pop_g(i,j+3)=rd;
            end
        end
    end
```

3.2 Application examples

In this subsection some application examples, using the algorithm described in the previous subsection, are presented. All examples are for binary images and use equations 1 and 2 shown in subsection 2.1. A synthetic image containing four objects with different shapes was generated for implementation of the examples. The training set consisted of three samples with different resolutions for each picture object. In addition, they were used three different maximum allowed sizes for each tree size, representing the chromosomes. Also, in each case the algorithm was run three times. In the first example, it was tried to find a combination of morphological filters and logical operators to recognize the star present in the input image (Figure 3). In this figure it is possible to see the desired image and the training process. The error found for this example was zero. Other training pairs were used through the training set also resulting in errors very close to zero. For this example the following genetic parameters and arguments were used.

ng=50,
nc=500,
tx_cs=90%,
tx_mt=10%,
arg=[1 2 3 4 5 6 7 8],
tb_arg=[0 0 0;0 0 1;0 1 0;0 1 1;1 0 0;1 0 1;1 1 0; 1 1 1].

The 'arg' vector pointers to 'tb_arg' that corresponds to the table of structuring elements used in the morphological operations. The instructions vector used in this example was the following:

instr={'nop' 'dil' 'ero' 'or1' 'and1' 'sto1' 'cpl'},

were:

nop means no operation;
dil corresponds to a dilation through the structuring elements contained in *tb_arg;*
ero corresponds to an erosion through the structuring elements contained in *tb_arg;*
or1 is equivalent to a logical OR operation;
and1 is equivalent to a logical AND operation;
sto1 corresponds to a storage operator of the current results in a temporary memory variable;
cpl is equivalent to a logical NOT operation.

"5 **4 2 4** 2 **7 5 2** 3 **5 3 6** 3 **2 2 8** 4 **4 5 6** 6 **6 7**
2 1 **2 4 5** 2 **4 4 4** 6 **6 3 7** 4 **6 3 6** 2 **7 5 7** 4 **4 7**
7 3 **7 3 2** 4 **6 7 3** 5 **2 4 5** 2 **2 3 6** 4 **5 7 2** 4 **7 5**
1 3 **2 7 3** 5 **5 5 7** 6 **6 6 1** 1 **2 8 8** 2 **5 7 6** 6 **5 5**
8 0 **6 6 5** 1 **6 5 2** 6 **1 2 2** 6 **4 3 3** 1 **5 2 7** 2 **7 2**
4 6 **6 2 3** 4 **5 1 2**"

Figure 4 shows the algorithm generated to recognize the star presented in the input image. The arguments are pointers to the arguments table containing the structuring elements

for the morphological operators. The generated machine code, in decimal, for the given example considering the architecture cited in this work can be seen as follows.
The above code is transferred to a FPGA dedicated processor architecture. The FPGA processor processes in real time the images from a video camera with the objective of determining the shapes intended, according to the training algorithm. The addresses for the arguments table containing the structuring elements are shown in bold characters, thus the three arguments will point to three rows of the table forming a structuring element of size 3x3, equivalent to the size used in implementing each stage of the above architecture. Using this approach it is possible to build 512 different shapes structuring elements. The numbers representing the functions are pointers to the instruction vector 'instr', whose index starts at zero, and are not represented in bold characters. Thus, for example, code 1 **6 5 2** corresponds to the instruction dil (img, [1 0 1, 1 0 0, 0 0 1]), which is a dilation of 'img' by the structuring element shown as argument. The image 'img' is obtained from the result of the previous instruction by a pipeline process, and so on.

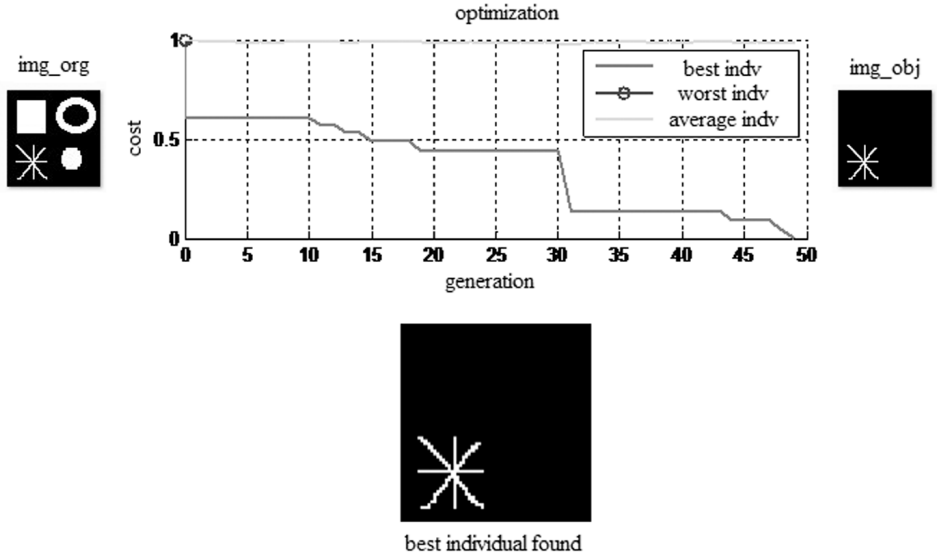

Fig. 3. Example of an automatic filter construction for recognizing the star present in the input image

Another application example obtained by the developed system, was the decomposition of an any shape structuring element in smaller size 3x3 structuring elements. For this example we used the same parameters of the previous example. The training process for the evolutionary system of the given problem is illustrated in Figure 5. A binary 17x17 size object, not decomposed by the algorithm proposed by Park and Chin (Park & Chin, 1995), could be decomposed by the proposed methodology. The process converged after iteration 300, and the error obtained was zero. The following opcode was generated for the addressed problem.

"0 7 2 5 0 1 2 4 0 5 1 2",

where the zeros correspond to successive dilations of the input image by the following structuring elements.

[1 1 0;0 0 1;1 0 0], [0 0 0;0 0 1;0 1 1], e [1 0 0;0 0 0;0 0 1]

After successive dilations of a central point by the above structuring elements, it was possible to find the object shown in Figure 5.

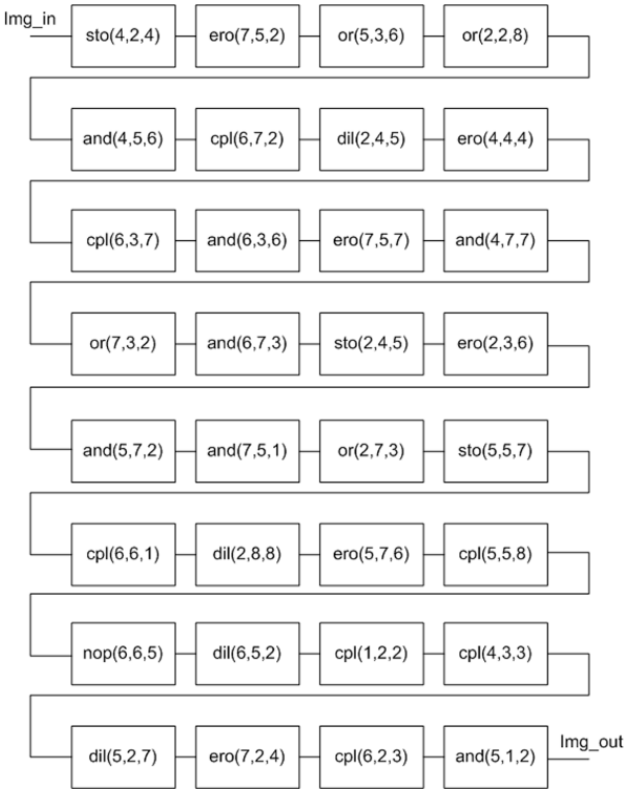

Fig. 4. Algorithm generated for the automatic pattern recognition filter construction problem shown in Figure 3

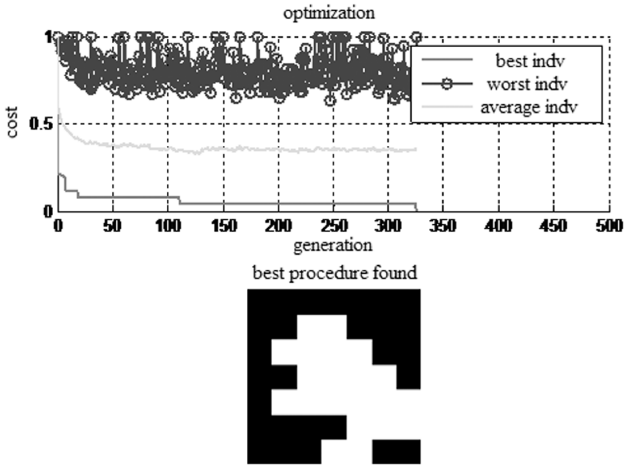

Fig. 5. Example of decomposition of a 17x17 structuring element

As mentioned earlier in this chapter, the results obtained by the aforesaid system were used to set up a FPGA implemented architecture, whose block diagram is shown in Figure 6. The architecture has several reconfigurable pipeline stages that can deal with 3x3 structuring elements. The concatenation of several stages allows operations with larger size structuring elements, whose shape can also be flexible.

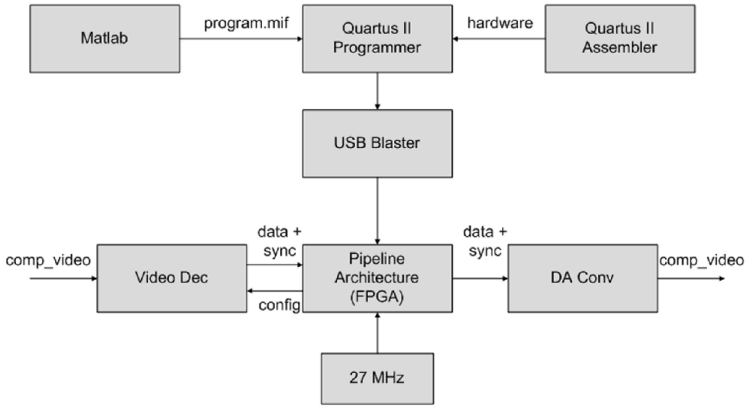

Fig. 6. Architecture used to process the programs generated by the proposed system (Pedrino et al., 2010)

4. Conclusion

The construction of a Matlab algorithm using a methodology for automatic construction of morphological and logical operators by the use of genetic programming was presented in this chapter. When presenting pairs of images to the system from a training set, a set of

instructions and arguments for a given problem and appropriate genetic parameters, an evolutionary process builds a sequence of nonlinear image operators that given an input image produces an output image as close as possible to the goal image provided. The algorithm generated in this work was also used to configure a pipeline processing architecture in FPGA, capable of processing images in real time, with the images provided by a CCD video camera. Examples were shown in the text in order to demonstrate the feasibility of the developed methodology for automatic construction of image processing algorithms. The task of designing an imaging processing sequence of operators is not so trivial, so the proposed methodology might be very helpful as an aid for the expert in this situation.

5. Acknowledgments

Emerson C. Pedrino is grateful to the "Fundação de Amparo a Pesquisa do Estado de São Paulo" for the financial support of this work, thoughout the project, proc. 2009/17736-4. The authors are also grateful to the Departament of Computer Science/University Federal de São Carlos, Faculty of Campo Limpo Paulista, and to the Department of Electrical Engineering/UFRN.

5. References

Dougherty, E. R. (1992). *An Introduction to Morphological Image Processing*, SPIE, Bellingham, Wash, USA

Serra, J. (1982). *Image Analysis and Mathematical Morphology*, Academic Press, San Diego, Calif, USA

Weeks Jr., A. R. (1996). *Fundamentals of Electronic Image Processing*, SPIE, Bellingham, Wash, USA

Soille, P. (1999). *Morphological Image Analysis, Principles and Applications*, Springer, Berlin, Germany

Sonka, M., Hlavac, V. & Boyle, R. (1993). *Image Processing, Analysis and Machine Vision*, Chapman & Hall, Boca Raton, Fla, USA

Facon, J. (1996). *Morfologia Matemática: Teoria e Exemplos*, Editora Universitária da Pontifícia Universidade Católica do Paraná, Prado Velho, Brazil (In portuguese)

Pedrino, E. C., Roda, V. O. & Saito, J. H. (2010). A Genetic Programming Approach to Reconfigure a Morphological Image Processing Architecture. *International Journal of Reconfigurable Computing*, Vol.2011, pp. 712494-712503

Gonzalez, R. C. & Woods, R. E. (2008). *Digital Image Processing*. Prentice Hall, Upper Saddle River, NJ

Angulo, J. & Serra, J. (2005). Morphological Coding of Color Images by Vector Connected Filters, In: *Centre de Morphologie Mathématique*, Ecole des Mines de Paris, Paris, France

Chanussot, J. & Lambert, P. (1998). Total ordering based on space filling curves for multivalued morphology, In: *Proceedings of the 4th International Symposium on Mathematical Morphology and Its Applications*, 51–58

Koza, J. (1992). *Genetic Programming*, MIT Press, Cambridge, Mass, USA

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, Mass, USA

Park, H. & Chin, R. T. Decomposition of arbitrarily shaped morphological structuring elements, *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol.17, No1, pp. 2-15

MATLAB - A Ubiquitous Tool for the Practical Engineer

Edited by Prof. Clara Ionescu

ISBN 978-953-307-907-3
Hard cover, 564 pages
Publisher InTech
Published online 13, October, 2011
Published in print edition October, 2011

A well-known statement says that the PID controller is the â€œbread and butterâ€ of the control engineer. This is indeed true, from a scientific standpoint. However, nowadays, in the era of computer science, when the paper and pencil have been replaced by the keyboard and the display of computers, one may equally say that MATLAB is the â€œbreadâ€ in the above statement. MATLAB has became a de facto tool for the modern system engineer. This book is written for both engineering students, as well as for practicing engineers. The wide range of applications in which MATLAB is the working framework, shows that it is a powerful, comprehensive and easy-to-use environment for performing technical computations. The book includes various excellent applications in which MATLAB is employed: from pure algebraic computations to data acquisition in real-life experiments, from control strategies to image processing algorithms, from graphical user interface design for educational purposes to Simulink embedded systems.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Emerson Carlos Pedrino, Valentin Obac Roda and Jose Hiroki Saito (2011). Non Linear Algorithm for Automatic Image Processing Applications in FPGAs, MATLAB - A Ubiquitous Tool for the Practical Engineer, Prof. Clara Ionescu (Ed.), ISBN: 978-953-307-907-3, InTech, Available from: http://www.intechopen.com/books/matlab-a-ubiquitous-tool-for-the-practical-engineer/non-linear-algorithm-for-automatic-image-processing-applications-in-fpgas

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821