Structural Description of the Retina with Soft Computing Methods

Gabriella Kókai

Dept. of Computer Science Friedrich-Alexander Univ. of Erlangen-Nürnberg
Martensstr. 3. D-91058 Erlangen, Germany
Phone: (49) +(9131) 8527830, Fax: (49) +(9131) 8528809
e-mail: kokai@informatik.uni-erlangen.de

Róbert Ványi, Zoltán Tóth

Institute of Informatics József Attila University
Árpád tér 2, H-6720 Szeged, Hungary
Phone: (36) +62 420 184, Fax: (36) +62 420 292
e-mail: h531774|h531714@stud.u-szeged.hu

Tünde Pető

Department of Opthalmology A. Szent-Györgyi Medical University
Korányi f. 12, H-6720 Szeged, Hungary
Phone: (36) +62 45 5487, Fax: (36) +62 45 4573
e-mail: peto@opth.szote.u-szeged.hu

Abstract

In this paper the *GREDEA* system is presented. The main idea behind it is that with the help of evolutionary algorithms a grammatical description of the blood circulation of the human retina can be inferred. The system uses parametric *Lindenmayer systems* as description language. It can be applied on patients with diabetes who need to be monitored over long periods of time.

Keywords: Computer vision, image and signal interpretation, Lindenmayer-systems, evolutionary algorithms

1 Introduction

In this paper the *Grammatical Retina Description with Evolutionary Algorithms* (*GREDEA*) system is presented. The main idea behind *GREDEA* was to develop

patient-specific monitoring programs for examining the blood circulation of the human retina. The system can be used on patients with diabetes who need to be monitored over long periods. It is essential to check the eyesight of patients with this disease, because the deterioration of the vascular tree caused by diabetes has a direct effect on the vision quality.

In recent years some systems (such as [5], [11] and [27]) have been developed but these systems employed computer graphic methods. They require comparison of different images taken in regular intervals. To make a diagnosis based on these images is a difficult task and needs a lot of expertise. Another attempt to use a syntactic approach is the system developed by Hesse et al. ([9]). Here fractal modelling was applied but the authors concentrated on simulating the typical growth pattern of the retinal artery and so this work has no real diagnostic benefit.

In *GREDEA fundus images* are used. In the first step these images are preprocessed, several image processing routines are applied. In the resulting image background is black (0 values) and the vascular tree is white (1 values). These images are later referred to as *destination images*.

A parametric *Lindenmayer system* (*L-system*) is developed which creates a pattern close to the vascular tree of the patient. This *L-system* gives an individual description of the blood circulation of the human retina and can be stored (needs less storage than storing a picture) and used later to make comparisons.

L-systems are parallel grammatical rewriting systems. They contain an initial axiom and one or more rules. With the help of such a grammar complex objects can be described starting from the axiom and using the productions [19]. When choosing the representation form, the fact that *L-systems* can be evolved using evolutionary algorithms [14] was taken into consideration.

The evolutionary algorithms can be briefly described as follows: a model of natural evolution [2] is often used as a way of solving optimization problems. These algorithms are based on the collective adaptation and learning ability of *individuals*. Individuals form a *population*, and each individual represents a possible solution of the problem. At the beginning, the individuals in the population are initialized randomly. They can be modified by the operators *selection*, *mutation* and *recombination*. These modifications lead to the evolution of the individuals; during the process better and better individuals appear. Thus, in later generations individuals that represent better solutions are more likely to appear.

Since parametric L-systems are evolved, two kinds of evolutionary algorithms are applied: *genetic algorithms* [6] are used on the rewriting rules of the evolved L-system, and *evolution strategies* [22] are applied on the parameters of the L-systems.

We proceed as follows. First in Section 2 and 3 a brief introduction to retina images and some details about image processing are given. Section 4 contains the detailed description of L-systems, evolutionary algorithms and the application of evolutionary algorithms on L-systems. The evolution process is described in Section 5. Implementation details can be found in Section 6. In Section 7 an example for the experiments is discussed. Finally, Section 8 contains a summary and some comments on future work.

2 The retina

The retina is the light sensitive surface at the back of the eye [15]. The retina contains a vast number of specialized nerve cells called rods and cones that convert light rays into nerve signals. The cones provide color vision, and are most con-

centrated at the fovea. The fovea can be seen in Figure 1 as the region on the side without any blood vessels crossing it. Because of their dense concentration, the cones in the fovea provide the finest resolution vision.

The rods are much more sensitive to light than the cones but they only provide black and white vision. All of the rods and cones connect with nerves whose signals exit from the eye via the optic nerve. The optic nerve along with the retinal artery joins the eye at the optic disk. The optic disk can be seen in Figure 2.

There are no rods or cones in the optic disk which creates a small blind spot on the retina. The vessel system of the eye background consists of two types of blood vessels, arteries and veins. The part where the vessels (arteries and veins) enter the eye is called papillary. Arteries and veins may have some cross points in the eye background, but arteries never cross arteries and veins never cross veins. The blood vessels can be described with a binary tree, because from a node a vessel can branch off only in two directions. The vessels starting from the papillary surround the macula. The macula does not contain any vessels.

When diabetes affects the eyes, it often affects the retina lining the back of the eye, specifically the small blood vessels (see in Figure 3) which nourish this membrane. This condition is called *diabetic retinopathy*. Diabetic retinopathy often has no symptoms in its early, most treatable stages. Improving awareness of the importance of early detection and treatment is imperative if one is to reduce visual loss from this disease. A minimal evaluation of the diabetic retina consists of regular examinations by a qualified practitioner through dilated pupils.

3 Processing retina images

In medicine several type of retina images are used. In *GREDEA* only fundus images are used. They are taken with a *fundus camera*. This machine uses prisms and lenses to take a regular photograph of the selected area. As a consequence of the structure of the eye, only a small part can be photographed at a time. Therefore the fundus camera can be moved not only horizontally and vertically, but also can be slanted to view up or down. Before the process the pupil in the patient's eye must be enlarged to get as sharp an image as possible. But still the iris shadows on the retina, so the side of the image is darker. The fovea is also darker, but as it was mentioned, the fovea usually does not contain blood vessels.

3.1 Structure of the images

In the fundus images blood vessels are not only darker than the background, but their color is red, while the background is orange. To detect the blood vessels, a gray-scaled image was used. Instead of creating a normal gray-scaled image, the original picture (Figure 4) was decomposed to three colour channels, namely to the red, the green, and the blue component. From these the green component (Figure 5) was used, because colour red does not contain a green component, but orange does. Next the histogram of the image is created (see Figure 6). According to this diagram three types of pixels can be distinguished. They are denoted with *a b* and *c*. The dark pixels (type *a*) are the circular frame of the image. The light pixels (type *c*) are from the papillary. The most important part is located between these and contains the pixels from the vessels and the background. Unfortunately they cannot be separated using only the histogram.

3.2 Edge detection

To separate the blood vessels from the background an edge detection algorithm is used, namely the Robert's gradient algorithm which works as follows.

Let I be a gray-scaled image and (x,y) an arbitrary pixel. The new $I'(x,y)$ value of this pixel is:

$$I'(x,y) = \sqrt{a^2+b^2}, \text{where}$$
$$a = I(x,y) - I(x+1,y+1) \text{ and } b = I(x,y+1) - I(x+1,y)$$

The resulting image was stretched so the lightest color is white, and thresholding were also applied to change dark pixels to black and light pixels to white. Finally the isolated pixels were filtered. A white pixel is called isolated, when none of its eight neighbors is white. The result can be seen in Figure 7 and 8. The first image is the preprocessed gray-scaled image, the latter is the preprocessed green component. In Figure 9 the difference can be seen so one can notify that using the green component more of the vascular tree can be found. For better quality and visibility in this paper these images are inverted, so background is white while the blood vessels are black. The whole image processing algorithm is the following:

First, create the green component:

for $y = 0$ **to** $height - 1$ **do**
  for $x = 0$ **to** $width - 1$ **do**
    $green_image[x,y] = image[x,y].green$

Next calculate Robert's gradient together with thresholding, and also the isolated pixel filtering.

for $y = 0$ **to** $height - 2$ **do begin**

  for $x = 0$ **to** $width - 2$ **do begin**

    $a := green_image[x,y] - green_image[x+1,y+1]$

    $b := green_image[x,y+1] - green_image[x+1,y]$

    $e :=$**sqrt**$(a*a+b*b)$

    if $e > THRESHOLD$

      then $dest_image[x,y] :=$white

      else $dest_image[x,y] :=$black

  end

end

for $y = 1$ **to** $height - 2$ **do begin**

  for $x = 1$ **to** $width - 2$ **do begin**

    if $dest_image[x,y]$=white **and** (x,y) is isolated

      else $dest_image[x,y] :=$black

  end

end

4 Fundamentals of the evolution using L-systems

In this section a short overview of concepts used in this paper is given. First the Lindenmayer system is defined. The turtle graphic draws the individuals generated by the grammar on the screen. Then a short informal description of genetic algorithms used to evolve the grammar rules is presented.

4.1 Lindenmayer systems

Lindenmayer introduced special kinds of descriptive grammars in order to describe the natural development processes of plants [16]. These are called Lindenmayer grammars or *Lindenmayer system*s (*L-system*s). The most important characteristic of *L-system*s compared with Chomsky grammars is *parallel rewriting*. In Lindenmayer systems rewriting rules are applied in parallel and simultaneously replace all letters in a given word.

The definition of a Lindenmayer system is the following: $G = (\Sigma, \pi, \alpha)$, where

- $\Sigma = \{s_1, s_2, ..., s_n\}$ is an alphabet,
- $\pi : \Sigma^+ \rightarrow \Sigma^*$ is a homomorphism, and
- $\alpha \in \Sigma^+$ is an axiom.

Thus, a Lindenmayer system can be characterized as follows:

- Description the initial state
- One or several rewriting rules: these determine how the object is developed
- Description of the current state: shows the attributes of the simulated object. This description is a series of state indicators created by modifications of the initial state. These modifications have to correspond to the rewriting rules. Depending on how many times the rewriting rules are applied, one can speak about *generation* $0, 1, \ldots, n$. Generation 0 is the initial state

Different *L-system* classes are introduced, such as deterministic or nondeterministic *L-systems*. The distinction can also be made between context free and context sensitive *L-systems*.

4.1.1 Context-free L-systems

An L-system is context-free if its state indicators do not affect the evolution of the state indicators neighbouring them. This is called $0L$*-system*. The 0 shows that the evolution of a state indicator depends only on the state indicators in distance 0 from it. Later similar tokens $(1L, 2L, (m,n)L)$ are used to mark context-sensitive L-systems. The simplest L-systems are the deterministic $0L$-systems ($D0L$-systems). In this paper only this kind of L-systems is studied.

4.1.2 D0L-systems

The most important characteristic of $D0L$-systems is that each state indicator has at most one rewriting rule. Thus, the evolution of the $D0L$-system is unambiguously determined. The mathematical definition of $D0L$-systems is similar to the one described above. Σ^* denotes the set of all finite sequences generated from the elements of the alphabet. Σ^+ denotes the $\Sigma^* - \lambda$, where λ is the empty sequence. If the rewriting rule for an element $s_i \in \Sigma$ is not determined $\pi(s_i) = s_i$.

The application of rewriting rules can be performed as follows: all sequences can be decomposed to $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$ parts. Thus $\pi(\varepsilon) = \pi(\varepsilon_1\varepsilon_2 \ldots \varepsilon_n)$ and if $\varepsilon_i = s_j$, then $\pi(\varepsilon_i) = \pi(s_j)$, otherwise ε_i can be broken into parts of sequences, too.

Other types of L-systems are the context-sensitive L-systems and the parametric L-systems. The most important characteristic of L-systems compared with Chomsky grammars is *parallel rewriting*. When the current string is rewritten, all the occurrences of each symbol staying on an left side of a rewriting rule are simultaneously replaced by appropriate the right side of the rule. In this paper only *deterministic context free L-systems* (*D0L-systems*) are used so it will be described in the next section. To enlarge the number of described objects, *L-systems* can also be associ-

ated with parameters.

4.1.3 Parametric D0L-systems

With the idea of parametric *L-systems* the concept of parallel rewriting to parametric words instead of strings of symbols can be applied. Following the notation of Prusinkiewicz [20] the word "module" as a synonym for "letter with associated parameters" is used.

The alphabet is denoted by Σ, and the set of parameters is the set of real numbers $\boldsymbol{R}$. A module with letter $a \in \Sigma$ and parameters $p_1, p_2, \ldots, p_n \in \boldsymbol{R}$ is denoted by $a(p_1, p_2, \ldots, p_n)$.

The real-valued *actual* parameters appearing in words have a counterpart in the *formal* parameters which may occur in the specification of *L-system* productions. Let Π be the set of formal parameters. The combination of formal parameters and numeric constants using the arithmetic operators $(+, -, *, /)$, the relational operators $(<, <=, >, >=, ==)$, the logical operators $(!, \&\&, ||)$ and parentheses $(())$ generates the set of all correctly constructed logical and arithmetic expressions with parameters from Π. Thet are denoted as $\mathcal{C}(\Pi)$ and $\mathcal{E}(\Pi)$.

Definition A *parametric D0L-system* is an ordered quadruple $G = (\Sigma, \Pi, \alpha, P)$, where

- $\Sigma = s_1, s_2, ..., s_n$ is the alphabet of the system,
- Π is the set of formal parameters,
- $\alpha \in (\Sigma \times \boldsymbol{R}^*)^+$ is a nonempty parametric word called the axiom,
- $P \subset (\Sigma \times \Pi^*) \times \mathcal{C}(\Pi) \times (\Sigma \times \mathcal{E}(\Pi)^*)^*$ is a finite set of productions.

A production in a *D0L-system* is given in the form $pred \mid cond \rightarrow succ$, where

pred $\in \Sigma \times \Pi^*$ denotes the *predecessor*, *succ* $\in ((\Sigma \times \mathcal{E}(\Pi)^*)^*$ is the *successor* and *cond* $\in \mathcal{C}(\Pi)$ describes the *conditions*. A production *matches* a module if the following conditions are met:

- the letter in the module and the letter in the production predecessor are the same
- the number of actual parameters in the module is equal to the number of formal parameters in the production predecessor
- the condition evaluates to true if the actual parameter values are substituted for the formal parameters in the production.

If a production matches a module then it can be *applied* in order to create a string of modules specified by the production successor. The actual parameter values are substituted for the formal parameters according to their position.

There is no essential difference between *D*0*L-systems* that operate on strings with or without brackets. In our implementation brackets are used. However, in this case productions involving brackets are restricted to the following forms:

- *pred* | *cond* $\rightarrow$ *succ*, where *pred* and *cond* are as before, but *succ* $\in ((\Sigma \times \mathcal{E}(\Pi)^* \cup \{[,]\})^*$, and *succ* is well nested;
- $[\rightarrow [$, or $] \rightarrow]$.

A string *S* is called well-nested if and only if for all S' prefix of *S* the number of '[' symbols is not less than the number of ']' symbols, and the number of these symbols is equal in *S*. More details and examples of parametric *D*0*L-systems* can be found in [21].

4.2 Displaying images defined by L-Systems

To display an image defined by an *L-system* the well-known turtle graphic is used [1]. The turtle is an object in space such that its position is defined by a vector and its orientation is defined by a coordinate system (see Figure 10). The coordinate system determines the heading ($\vec{h}$), the left ($\vec{l}$), and the up ($\vec{u}$) directions. Using these vectors the turtle can be moved in space. There are several turtle operations, also known as turtle commands. The most important ones are drawing ('F') and turning left and right ('$+$' and '$-$'). There are special turtle operations, called stack operations. These are denoted by '[' and ']', and used to save and restore turtle position and orientation. Using stack operations branching structures can be produced. The used turtle commands are listed in the Appendix.

To use turtle graphics with *L-system*, turtle commands as the elements of the alphabet are applied.

4.3 Evolutionary algorithms

Evolutionary algorithms are computer-based problem solving systems which use evolution processes as they can be observed in nature ([2]). Main types of evolutionary algorithms are evolutionary strategies ([22], [24]), genetic algorithms ([6], [10]) and genetic programming ([14]). All of them are based on the simulation of evolution of individuals by means of natural *selection*, *mutation* and *reproduction*.

More exactly, evolutionary algorithms work with *populations of structures* which are evolved by using *selection*, *mutation* and *reproduction*. Each individual is provided with a *fitness value*, which shows how the individual solves the problem. Reproduction selects individuals with high fitness values, and recombination and

mutation work on these individuals. Thus, the preservation of good attributes is assured.

4.3.1 Genetic programming

A genetic algorithm is a model of a machine which performs operations of natural evolution handling a *population whose* individuals are represented by *chromosomes*. The representation form of individuals in genetic algorithms is usually a bitstring. Genetic programming is a type of genetic algorithm, where chromosomes with unspecified length are used. It means that the size of the chromosomes can be dynamically changed during the process.

It can be noticed that evolution in nature is not a guided process. In natural processes individuals race for environmental resources. Some are better than the others. The better ones are more likely to survive, so they can reproduce their good attributes to their descendants.

In molecular level the pairs of chromosomes collide, exchange their parts, and then leave each other. This operation is called *crossover*. When crossing two individuals, first the parent individuals must be selected. After that one or more crossover points are established, and the corresponding parts of the chromosomes are exchanged. There is a crossover method which is not based on the processes observed in nature – *uniform crossover*. This method produces only one individual in the way that at every bit a random number determines whether to take that bit from the first or the second parent. The individuals to cross during genetic algorithms are selected on the basis of their fitness value.

Mutation also operates in the algorithm, but its importance is subject of discussions. Mutation means flipping bits in the chromosomes.

When implementing the basic genetic algorithm, usually the following loop is used:

- The fitness value of each individual in the population is determined
- A new population is generated by applying the genetic operators of recombination, mutation and some kind of selection
- Consider the new population as the actual population and continue the process with the next iteration

One iteration of the process described above is called *generation*. The first (0.) generation works with a population of randomly created individuals. After that, genetic operators work according to the individuals' fitness values, and so the individuals show better and better behavior.

4.3.2 Evolutionary strategies

Evolutionary strategies (ESs) were introduced in the 1960s by Rechenberg ([22]) working in Berlin and further developed by Schwefel ([24]).

The main difference between genetic algorithms and evolutionary strategies (ESs) is that the representation form of the individuals is usually real vectors. Thus, genetic operators (crossover – which is called recombination here – and mutation) are different.

At recombination, it is possible to use more than two parents to create one descendant. There are two kinds of recombination: *discrete recombination* is like uniform crossover in GAs, i.e. each real number in the vector chromosome is selected randomly from one randomly selected parent. When using *intermediate recombination*, the vector elements for the descendant are computed by averaging the corresponding elements of the parent vectors.

Mutation is done by adding a randomly generated $N(0,\sigma)$ normal distribution number to the vector elements, where σ is dependent on the domain of definition of the vector elements.

There are two kinds of evolutionary strategies: $(\mu/\rho+\lambda)$-ES (plus strategy) and $(\mu/\rho,\lambda)$-ES (comma strategy). In both cases, μ, ρ and λ denote the size of the population, the number of parents and the number of descendants to produce, respectively. At *plus strategy*, first λ individuals are created, and then the best μ individuals are selected to survive from the existing $\mu+\lambda$ individuals. At *comma strategy*, after creating the λ descendants, the parents die off, and then the surviving μ individuals are selected from only the newly generated individuals.

5 The evolution process

If an image is given and one wants to find the grammar that describes it, this is called the inverse problem of L-systems. Evolutionary algorithms used to solve this problem are described here in detail.

5.1 Initial population

The first step is to define the structure of the initial population and the individuals themselves. For the first examination the initial population is created randomly, but if the patient returns for a checkup, the previously generated L-system is used to create the initial population. This increases the convergence speed and makes comparison easier.

As was mentioned before, the individuals are represented by *parametric L-systems*. According to the definition, parameters and constant values may appear in the

rewriting rules. The *constant values* are considered as the *parameters* of the whole L-system. For example, if an $X \rightarrow Y(10)$ is present, it is assumed that an $X \rightarrow Y(A)$ rule is used and the L-system has a parameter called 'A' with a current value of 10. It is also assumed that the axiom is S.

Because the shape of the generated image depends both on the rewriting rules and on the parameters, the representation can be divided into two important parts. The right sides of the rules are represented as strings, while the parameters are described as vectors of real values. Usually it is more important to find the appropriate right sides. Changing the parameters is used for the fine-tuning of the image.

L-systems with the same rules are considered as an *L-system group*. In an *L-system group* the rules are constant, but the parameters can be varied. To find the best parameters *evolutionary strategies* (*ES*) are used inside the *L-system group*. Therefore an *L-system* group works as a *subpopulation*. A subpopulation can be represented by rewriting rules. Because the number of the rules, and also the left sides are constant, an *L-system group* can be represented by an array of strings. To find the rewriting rules, a *genetic algorithm* (*GA*) is applied on the string arrays representing an *L-system group*. So the *L-system groups* make up the population. The structure of the population can be seen in Figure 11.

5.1.1 The structure of the population

To calculate the fitness value, the images described by the L-systems are created, then the distance between these generated images and the destination image is calculated with a suitable function. This function will be discussed in section 5.2. Using this fitness value *evolutionary strategies* are applied on the parameter vectors several times. The number of the evolution steps (denoted by S_{ES}) is defined at the very beginning of the process. When the ES steps are finished, the best individuals

in each subpopulation are selected. The parameters and the fitness values of these *L-systems* will act as the parameters and the fitness value for the whole subpopulation. Using this fitness value, several subpopulations can be selected and can then be used to create new subpopulations with *genetic algorithms* (*GA*). The detailed description of the applied ES and GA operators can be found later in section 5.3. Earlier it was mentioned that ES is usually applied on real valued vectors, however GA is used with words over a finite alphabet. Therefore in this process both of them are used.

5.1.2 The Form of an Individual

The base *L-system* used as an individual is the following:

alphabet: $\Sigma = \{S, F, +, -, [,]\}$

axiom: $\alpha = S$

rewriting rule: $S \to u$, where $u \in \Sigma^*$ is a well-nested string.

Using this approach it can be noticed that this type of *L-system* sometimes generates *unnecessarily long word*s (*ul-words*). For example the image described by the word $F+--++F-+$ can be also generated by the word $F+F$, which is much shorter. To solve this problem, sequences of symbols can be defined, such that these sequences always start with a drawing symbol and do not contain unnecessary turtle movements (e.g. $+--+$). If *regulated evolution* is used to *preserve sequences* those ul-words can be avoided [12]. During regulated evolution, sequences cannot be broken, so genetic operators must be applied on substrings and not on single symbols. This makes the genetic algorithm more difficult. The other solution employs the specialties of the problem. By examining the blood vessels in the eye, a better *L-system* can be defined which itself includes the sequences:

$\Sigma = \{S, F, +, -, [,], B, Y, T\}$

$\alpha = S$

rewriting rules:

- $S \to v$, where $v \in \{S, B, Y, T, [,]\}^*$ is a well-nested string.
- $B \to S{+}F$
- $Y \to F[+S][-S]$
- $T \to F[+S]S$

Note that the vascular tree can contain only the following parts: *bending* (B), *branching* (Y) and *forking* (T). Using this *L-system*, only the first rule will be modified during the evolution process. The initial F symbols of the other three rules ensure that ul-words will not be created. So this *L-system* will produce better words, just like the *L-system* with sequences, without making the use of genetic operators difficult.

The turtle commands F, $+$ and $-$ all have one parameter to define the length of the line or the angle of rotation respectively. To append these parameters, the L-system defined above must be slightly modified. So the form of an individual is the following *parametric L-system*:

$\Sigma = \{S, F(a), +(a), -(a), [,], B(a,b), Y(a,b,c), T(a,b)\}$

$\alpha = S$

rewriting rules:

- $S \to v$, where $v \in \{S, B(a,b), Y(a,b,c), T(a,b), [,]\}^*$ is a well-nested string, with constant parameters.
- $B(a,b) \to S{+}(a)F(b)$
- $Y(a,b,c) \to F(a)[+(b)S][-(c)S]$
- $T(a,b) \to F(a)[+(b)S]S$

The form of the first rule is like the following:

$$S \to B(A,B)Y(C,D,E)Y(F,G,H)T(I,J)B(K,L).$$

In this case the L-system has twelve parameters: from 'A' to 'L'.

5.2 The fitness function: the distance of two images

There are many possibilities to define the distance between two images. A very simple method to calculate the distance d_q between two images I_1 and I_2 is the quadratic error of the images. During the definition it is assumed that the sizes of both pictures are the same: $N \times M$.

The distance is the sum of quadratic errors of two corresponding pixels:

$$d_q(I_1,I_2) = \sum_{x=0}^{N-1}\sum_{y=0}^{M-1}(I_1(x,y) - I_2(x,y))^2 \tag{1}$$

For binary images, where $I_1^B(x,y)$ and $I_2^B(x,y)$ are two pixels the quadratic error is:

$$d_p(I_1^B(x,y),I_2^B(x,y)) = \begin{cases} 1 & \text{if } I_1^B(x,y) \neq I_2^B(x,y) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

So let the distance between two binary pictures be defined like this:

$$d_q(I_1,I_2) = \sum_{x=0}^{N-1}\sum_{y=0}^{M-1} d_p(x,y) \tag{3}$$

In this paper an extended version of this definition is used because in the evolution process described here a distinction was made between the following four cases for an arbitrary (x,y) point:

– The point (x,y) is 0 on both images, it is usually not important for calculating the fitness.