Concurrent Application of Genetic Algorithm in Pattern Recognition

Mario Köppen[1], Evgenia Dimitriadou[2]
[1]*Fraunhofer IPK*
Pascalstr. 8-9, 10587 Berlin, Germany
E-mail: mario.koeppen@ipk.fhg.de
[2] *Institut für Statistik und Wahrscheinlichkeitstheorie*
Technische Universität Wien
Wiedener Hauptstr. 8-10, A-1040 Wien, Austria
E-mail: evgenia.dimitriadou@ci.tuwien.ac.at

Abstract. *This paper proposes the fitness control procedure for the application of genetic algorithms (GA) in pattern recognition. Instead of using GA to solve a concomitant optimization problem, the task is rather represented as a set of data bitstrings, and a similarity measure among those data bitstrings. Then, the GA is used to select the subgroup yielding highest mutual similarity. The paper also provides and discusses some examples for such a procedure, taken from the fields: pattern classification, clustering, and measuring.*

1 Introduction

Many approaches to pattern recognition are related to an appropriate concept of pattern similarity, be it in classification, clustering, segmentation or verification. Since similarity is a non-crisp feature, it is reasonable to describe similarity by degree values. Then, including similarity into the analysis needs an objective function (efficient procedure) that evaluates to a degree of similarity or incorporates degrees of similarity into another computation. However, at the same moment the objective function entails the set of attainable solutions as well.

Looking more closely on similarity, it can be seen that similarity is related to localization of a set of data vectors in a (often metric) space of sufficiently high dimensionality. In such a context, we are looking for representing degrees of similarity by more than one numerical value. The approach followed in the work presented here is to derive a set of bitstrings reflecting the similarity of the given data and to restrict all further evaluations on the similarity between these bitstrings. The paper will detail on approaches to attain bitstrings that represent similarity, but it may be stated here that there are roughly spoken two fundamental ways: either the bitstrings are derived from vector data, by assigning a bitstring of length n to a vector of n components (so one bit represents one data component), or by applying a set of computations to the given data, each one producing its output as a bitstring (with one bit representing one data vector).

Once having such a bitstring representation of similarity, genetic algorithms (GA) can be used for further analysis [1]. It is obvious that similarity within a set of bitstrings is directly

related to building blocks, i.e. to positional correspondences between substrings of the bitstring occurring with higher frequency. In this context, the Schemata theorem gives the basic theoretical underpinning of the functionality of genetic algorithms. It indicates, with some restrictions, the accumulation of building blocks correlating to higher fitness in the evolving population. However, while useful for theory, schemata are rarely considered for the applications of GA itself. This also holds for image processing and pattern recognition. In [2], there is a comprehensive survey of applications of GA in image processing. It can be easily seen that all of the approaches use GA for solving optimization problems that come up during solving an image processing task by providing a suitable fitness function, an encoding scheme, and possibly by modifying genetic operators, and then selecting the best individual according to the given fitness measure.

So, the approach to use GA for quantifying similarity by counting on the ability to accumulate building blocks is rather new application of using GA in pattern recognition. The basic concept is illustrated in fig. 1. On the left hand side, there is what we consider the standard application framework of GA in pattern recognition: an optimization problem is defined and GA is used to evolve a good solution to that problem. Of course, this approach is not restricted to pattern recognition. All other optimization procedures (gradient descent, simulated annealing, dynamic programming,...) might be used as well. Usually, the advantage of GA is related to the complexity or non-linearity of the optimization task.

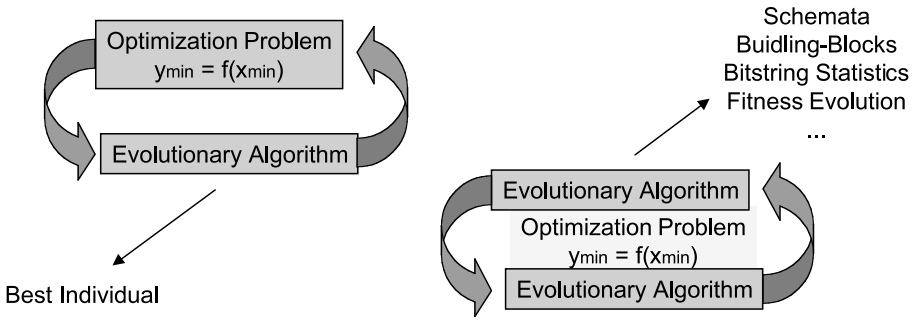

Figure 1: Concurrent approaches for using genetic algorithm in pattern recognition.

The right hand side of figure 1 gives the alternate procedure: while still providing an optimization task, its optimal solution is not the desired goal of the approach. Other information obtained from the evolving or from the evolved population will give the desired result. We will refer to this concurrent approach as *fitness control.*

The basic means for such evaluations will be given in the following sections, each of which is illustrated by an application example. But before, the underlying bitstring similarity problem has to be clearly stated and analyzed (section 2).

2 Generalized Bitstring Prototype Problem

This section will introduce the Generalized Bitstring Prototype Problem (GBPP). Examples of how this problem appears within the circumstances of pattern recognition problems will be considered in the following sections.

We begin with some explanatory remarks. The general situation is assumed as follows: given a set B of k bitstrings of length n each (i.e. sequence of n signs being either 0 or 1) that consists of $k_b \leq k$ repetitions of a bitstring b and the remaining bitstrings being randomly

distributed (random in the sense that it is equally likely for any bitstring element to be 0 or 1). We are in search for a procedure to find b from all bitstrings in B.

The basic procedure to solve this problem is the so-called *Hamming fusion* of all bitstrings: with $B = \{b_i\}$ and $b_i[j]$ being the sign at position j in bitstring b_i, $H_j^0(B)$ annotates the number of occurrences of a 0 at position j:

$$H_j^0(B) = |\{b_i \in B|\ b_i[j] = 0\}| \tag{1}$$

and $H_j^1(B) = |B| - H_j^0(B)$ gives the corresponding number of 1 at position j. Now, the Hamming fusion of B is given with the bitstring $t = HF(B)$ of length n and with $t[j] = 0$ if $H_j^0(B) > H_j^1(B)$, with $t[j] = 1$ if $H_j^0(B) < H_j^1(B)$ and either 0 or 1 if $H_j^0(B) = H_j^1(B)$. Obviously, $HF(B)$ is the bitstring with the smallest average Hamming distance to each bitstring in B (therefore we will refer to this operation as Hamming fusion).

Moreover, it may be expected that $HF(B) = b$ for the problem given above, since in case $b[j] = 1$ it may be estimated that $H_j^1(B) = (k - k_b)/2 + k_b > H_j^0(B) = (k - k_b)/2$ thus $HF(B)[j] = 1$ and similarily $HF(B)[j] = 0$ in case $b[j] = 0$.

An extension of the problem might be that there are not k_b identical copies of b in B but that these subsets are slight variations of b. As long as more than half of the bits at position j in this subset are equal to $b[j]$, the argument just given still holds, and the Hamming fusion will compute to b.

Thus, the problem of finding a subset of a set B of bitstrings, showing higher similarity of the bitstrings to each other than to the remainder of the set, can be replaced by the problem of finding a bitstring with lowest average Hamming distance to each bitstring to the whole set. Obviously, this is a feature of the Hamming distance metric that does not hold for other metrics like the euclidian distance.

2.1 Problem statement

Let $\pi = (a_1, a_2, \ldots, a_l)$ be a partitioning scheme of a bitstring of length $\sum_i a_i = n$. Thus, π describes a partition of any bitstring of length n into substrings of length a_i. Further be the symbolic Hamming distance between two such substrings of the same length given with 0 if both substrings are equal, and 1 if they differ in at least one position. So, the Hamming distance h_π (according to π) of the two (partitioned) bitstrings 0100011 and 0101010 is 1 for $\pi = (2, 1, 4)$ and 2 for $\pi = (1, 1, 1, 1, 1, 1, 1)$ (the conventional Hamming distance).

Note that the Hamming fusion can be easily extended to the case of general partitions π. Instead of counting frequencies of 1 or 0 at a bit position, the frequency of each substring is counted, and the substring (or any one of the substrings) with highest frequency gives the corresponding substring of the resulting bitstring. This Hamming fusion also serves the bitstring with smallest average Hamming distance according to π.

Also be $p > 0$ an integer, $cond$ a logical condition assigning $true$ or $false$ to a bitstring of length n, which selects the subset B_{cond} of bitstrings out of B that fulfill $cond$, and $crit$ an objective that can be computed from a set of p bitstrings (e.g. the sum of all pairwise Hamming distances to be maximal).

Then, a Generalized Bitstring Prototype Problem is given by the tuple $(\pi, cond, crit, p)$. The objective is for any given set B of k bitstrings of length n to find p bitstrings t_i of length n fulfilling

	00	01	10	11
00	0	1	1	2
01	1	0	2	1
10	1	2	0	1
11	2	1	1	0

Table 1: Hamming distances for $n = 2$.

$$\sum_{b \in B_{cond}} min_{i=1,\ldots,p} h_\pi(t_i, b) = min \quad (2)$$

$$crit[t_i] = max \quad (3)$$

Thus, the set of all t_i comprises a set of prototypes, to which the "joint" distance of each bitstring in B is minimal on average. As already noted, the Hamming fusion of B solves the simplest case, where π gives the bitwise Hamming distance, $cond$ always assigns $true$, $crit$ is constant and $p = 1$. In case $p > 1$, each prototype of the optimal choice will select (by smallest Hamming distance) a subset of B, of which the prototype itself is the Hamming fusion.

2.2 Discussion of simple cases

Next simplest case of the GBPP is the case $p = 2$. Here, no general solution can be provided, but some insight may be gained from considering special cases. Table 1 shows pairwise Hamming distances between all possible bitstrings of length 2. It can be seen that the minimum Hamming distance between any pair of different bitstrings to all four possible bitstrings is always the distance 0 in two cases and the distance 1 in the remaining two cases . Moreover, to each selection of the 0 and 1-assignments (two times each), a corresponding pair of bitstrings can be found so that their minimum pair distances just give these values. For example, to have pairwise Hamming distances 0 to the bitstrings 00 and 01, and 1 to the bitstrings 01 and 11, the pair has to be selected as $(00, 01)$. From this we may see the optimal choice for the bitstring pair. If bitstring b appears k_b times in B, the smallest achievable sum of pairwise minimum Hamming distances is obtained if and only if the distances 1 are assigned to the two smallest k_b. *Thus, the optimum selection are the two most frequent bitstrings in* B.

Having bitstrings of length $n = 3$ changes the situation. Table 2 shows Hamming distances between all eight possible bitstrings of length 3. It can be seen that the minimum between two columns is Pareto minimal if the columns corresponds to bitstring b and its inverse. In this case, the pairwise minimum distance is 0 in two cases, 1 in the remaining six cases. For all other pairs of bitstrings, each pairwise minimum distance is at least as large as in the case of inverse bitstrings.

Thus, when each possible bitstring of length 3 is present in B the sum of the pairwise minimal Hamming distances can not be smaller than for the pair of bitstrings $(b, \bar{b})$ for which $k_b + k_{\bar{b}}$ becomes maximum (thus minimizing the sum of all other frequencies biased with pairwise mimimum Hamming distance 1).

So, in case $n = 3$ with $k_b > 0$ for any b, the optimum choice of a pair of bitstrings (b_1, b_2) has the property that $b_2 = \bar{b_1}$ (this was not the case for $n = 2$).

	000	001	010	011	100	101	110	111
000	0	1	1	2	1	2	2	3
001	1	0	2	1	2	1	3	2
010	1	2	0	1	2	3	1	2
011	2	1	1	0	3	2	2	1
100	1	2	2	3	0	1	1	2
101	2	1	3	2	1	0	2	1
110	2	3	1	2	1	2	0	1
111	3	2	2	1	2	1	1	0

Table 2: Hamming distances for $n = 3$.

If some of the $k_b = 0$, this does not hold. Any pair can become optimal, as can be easily seen when composing B out of any two bitstrings only (which will comprise the optimal solution then as well).

For $n \geq 4$, the difference between odd and even n continues. Without proof, the following relation shall be noted here: Be $v(a, b)$ the vector obtained from the pairwise minimum Hamming distances of (a, b) to all bitstrings of length n (in a fixed order), i.e. for all bitstrings c of length n the smaller value of $h(a, c)$ and $h(b, c)$. Then, for $(b, \bar{b})$ any pair of inverse bitstrings, and (b_1, b_2) any pair of bitstrings such that in b_2 exactly $(n - 1)$ positions are inverse to the corresponding positions in b_1 (or $h(b_1, b_2) = n - 1$), the corresponding vectors $v(b, \bar{b})$ is a permutation of $v(b_1, b_2)$ if n is even.

This has an important meaning: if n is even and B consists of all possible 2^n possible bitstrings, each one occurring with equal frequency, then each pair (b_1, b_2) gives the optimum for the pairwise minimum Hamming distance if and only if b_1 and b_2 do not have more than 1 position in common. In case of odd n, b_1 and b_2 have to be inverse to each other.

However, the larger n the more unlikely it will become to have each $k_b > 0$ in B. And no closed solution can be provided here in case the k_b differ. So far, best known approach is to compare the sums of all k_b values for the Pareto set of pairwise mimimum Hamming distances of any two bitstrings. So, a similar study of the case $\pi = (2, 1)$ for $p = 2$ gives the size of this Pareto front to be 32 (out of 64 choices for the pairs of bitstrings).

To summarize, the solution of an GBPP strongly depends on the length n of the bitstrings, and the distribution of bitstrings in the set B. In case $p = 2$ a good heuristic is to check for pairs of inverse bitstrings. But in general, there are cases where the number of optimal solutions is exponentially increasing with n.

This discussion shows the high complexity of the problem even for the most simple cases. Many questions have to be left open here, and further studies may reveal new insights into the GBPP. At the present stage, no algorithm is known to reduce the exponentially growing number of choices that have to be tested for finding the minimum selection of prototype bitstrings.

2.3 Fitness control procedure

As stated in the foregoing section, the simple GBPP can be solved by directly computing the Hamming fusion of the set of bitstrings. In more complex cases, the ability of the GA to accumulate above-average schemata in the population might be of interest. Figure 2 demonstrates

how this can be also understood as a selection procedure. The left hand side of fig. 2 shows a unit square, with two marked areas within, so-called "manna." Each black dot represents an individual of a GA population, with (x, y) coordinates as their encoding, and a fitness function related to the presence of manna at the respective (x, y) position of the individual[1]. The right hand side gives the position of those individuals past ten generations. What can be easily seen is that most of the population now resides in the larger manna area. In other words: the ratio of individuals in the area does not reflect the ratio of the areas itself. It does reflect the ranking of the areas, or it can be considered that the GA did *select* the largest manna area.

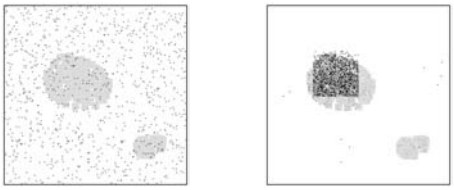

Figure 2: Genetic algorithm solving a selection task.

This gives an intuitive notation for the here-proposed GA application approach itself: given a set M of data m_i and a measure of similarity among data $s(m_i, m_j)$, a GA is able to select the largest subgroup of similar data. In the study so far, the data are supposed to be represented as bitstrings. For preventing mismatch with the bitstrings, which compose the population of a GA, we will refer to them as data bitstrings.

Then, the general fitness control procedure is as follows:

1. Design a set of data bitstrings from the application.
2. Define a GBPP for the data bitstrings, which gives a fitness function.
3. Let the GA evolve towards the minimal value of this fitness measure.
4. From the best individual p of the GA population, select all data bitstrings m_i with $s(p, m_i) < \theta$ with θ being a chosen threshold.

3 Application examples

3.1 Main color extraction

A simple application example is given by the task of detecting a dominant color in an image. This information might be of relevance for object extraction or image classification. Here, it is considered how bitstring fusion can be used to approach this task. In such a case, the image shall be given in a color model, e.g. in RGB represenation, with 8bit color depth for each of the three channels. Then, the binary representations of each color value are taken as bitstring, and the set of the bitstrings of all pixel in the image gives the set B. It has to be noted that the higher-order bits of each channel represent more of the color image information than the lower bits (e.g. with the first bit of a channel indicating whether the value at that position is larger than 128 or not).

[1] See http://www.caplet.com/MannaMouse.html for a detailed description of the experiment.

So, we consider the Hamming fusion of all pixel color values according to a partition π that handles the higher bits different than the lower bits in each channel. Examples are $\pi_1 = (2, 6, 2, 6, 2, 6)$ for a 24bit RGB color value, as well as $\pi_2 = (4, 4, 4, 4, 4, 4)$. Fig. 3 shows the output of these two fusions. In the subpictures (b) and (c), all pixel from the original image (a) were reproduced that were closer to the Hamming fused bitstring than 70. It can be seen that π_1 and π_2 behave different. So, while in the first 2 bit of each color channel the color green is more present, in the first 4 bit the red part becomes more dominant due to its higher homogenity (i.e. in the green parts are more variations of the bits 3 and 4). This shows how such a procedure can be adapted to various circumstances in main color selection.

(a) (b) (c)

Figure 3: Main color extraction due to different bitstring partitions: (a) original image with green and red pepper, (b) separation with $\pi = (2, 6)$ in each channel extracting the more inhomogenuous green parts, (c) using $\pi = (4, 4)$ extracts the more homogenuous red parts.

The reason to fuse the lower bits as well is that in this case there is a highly homogenous subpart of the image present, the fusion may give the corresponding color value more precisely.

So, the direct solution of the GBPP already gives very good results. Of course, the procedure can be applied in an iterative manner (e.g. to extract ink and paper color on document images). First, main color is extracted by fusing all color values, then the pixel similar to this main color are masked, and the main color of the remaining pixel is determined. Then, the procedure is repeated with the remaining pixels.

In this context, GA is not needed to obtain good results. However, in case two or more colors are expected to be present in the image to comparable degree, GA can be used to solve the corresponding GBPP.

3.2 Invoice table detection

An application for such a procedure was given in [3], in the context of invoice table detection.

The task was to find the table area in the digitized document of an invoice sheet. By applying operations of mathematical morphology, all text rows may be selected (this will not be detailed here). However, only a subset of those text rows compose the table. All rows belonging to the table are similarily formatted, but they are not completely equal. So, the table is defined as that subset of the set of all text rows, with higher pairwise similarity than to all other text rows.

Similarity may be measured by constructing templates from text rows. For doing so, the image of a text row is divided into cells, with cell width being a constant corresponding

to the estimated width of a character in a line of text. Now, a data bitstring is constructed from each text row, with the same size as the number of cells of a text row. Bitvalue 1 in the bitstring is set if the corresponding text row cell has more black than white pixels (thus, there is a character of text within), and bitvalue 0 otherwise. This ends step 1 of the given approach. In the further development it came out that the procedure was more reliable when using $p = 2$ prototype bitstrings, thus accounting for the high variations in the length of the article name entry. The GBPP formulated from this uses $\pi = (1, 1, \ldots, 1)$, $cond$ always true, $crit$ as constant and $p = 2$. After processing steps 3 and 4, the best individual has selected the text rows that compose the invoice table.

In [3], it was reported that such a procedure made only 23 errors for the evaluation of 184 table rows of 12 invoice sheets. Due to low computational demand of the fitness function calculation, this approach works in real-time. Further postprocessing might reduce that error rate even more.

3.3 Robust Clustering

Fitness control can be also applied to obtain robust (and blind) clustering of data. The basic assumption on robust clustering, which is employed here, is as follows: When a clustering method is inappropriate for the given data set, it will give more differing clustering results when the initial conditions or the given data values are randomly modified, than for an appropriate clustering method (see fig. 4).

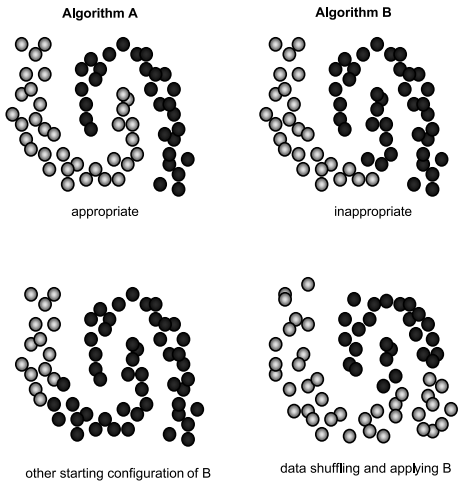

Figure 4: Applying different clustering methods to the same data.

Assume a (numbered) data set $d = (d_1, d_2, \ldots, d_n)$, which has to be separated into two clusters C_1 and C_2. Further assume a set of $M_1, M_2, \ldots, M_k$ clustering algorithms (e.g. k-means, neural gas,...), the $M_1(c_i), M_2(c_i), \ldots, M_k(c_i), 1 \leq i \leq 2$, results of which, when applied to d, will depend on an initial configuration of the c_i centers, and a set V of l random modifications of d, e.g. moving the data point by a small amount in a random direction. Then, when a clustering algorithm $M_p, 1 \leq p \leq k$, is applied to the modified data set $V_j(d)$, it will assign each data point in d to either C_1 or C_2. This can be represented by a data bitstring. For example, if there are four data points, the bitstring 0110 describes that data point d_1 and d_4 have been assigned to cluster C_1 and data points d_2 and d_3 have been assigned to cluster C_2. Applying all k clustering algorithms to all l modifications of the data set d will give

$k \cdot l$ bitstrings. Under the assumption given above, for appropriate clustering methods, there should be more similar bitstrings than for inappropriate ones. In other words: the common schemata of all $k \cdot l$ bitstrings hint on a more fitted clustering.

Again, steps 1 to 4 of the fitness control procedure can be performed. At the end, some of the data bitstrings are selected, yielding information by itself (as e.g. for the most suited cluster procedure among the n clustering algorithms).

The GBPP here was as before, with the $crit$ set to have maximum Hamming distance between the two prototypes. The reason for this is quite simple: all cluster variations yield two classes of output, but it can not be foreseen whether cluster 1 is always assigned to 1 or to 0. So, even completely equal clusterings may give bitstrings that are inverse to each other. With the additional criteria, the optimization may respect this issue.

The Intertwined Spirals data set [4] was analyzed this way [5]. At the end, the decomposition of the data set into the two spirals was found, and the Agglomerative Clustering also was found to be the most suitable procedure.

3.4 Heuristic measures

Instead of schematas, the fitness value obtained after a fixed number of GA generations might be of interest as well. Thus, problem classes can be defined according to intervals, into which such a fitness value falls.

A more sophisiticated example for such a "fitness calibration" was given in [6]. There, the issue of KIRLIAN images was considered, and the task was to study whether KIRLIAN images reflect some kind of intra-person specificity or not. For doing so, KIRLIAN images from all fingers were taken from a number of subjects at different times, and subjects were selected with perceptually similar KIRLIAN patterns. Now, a GA was used to *measure* that perceptual similarity.

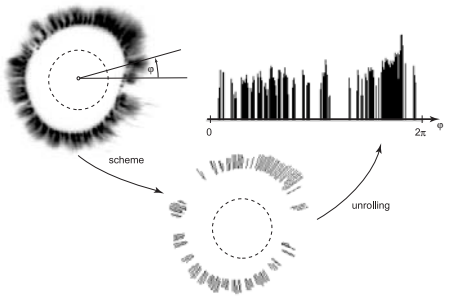

Figure 5: Enrolling of a KIRLIAN image.

For doing so, each finger image was unrolled first (see fig. 5). The unrollings were used to define the data bitstrings: if at angle ϕ the fraction of the beam in the direction ϕ and starting from the (interactively set) center C of the KIRLIAN image, which crosses the pattern is larger than 25 per cent of the maximum value in any direction, the bit was set to 1, 0 otherwise.

Each group of perceptual similar KIRLIAN images gives a set of data bitstrings, and we may start steps 2 and 3 of the fitness control procedure. However, step 4 is now replaced by taking the best fitness value after 200 generations.

The experiment was repeated with groups of randomly selected KIRLIAN images (i.e. images of different subjects), and also with random settings for the unrollings.

The obtained fitness values belonged to three classes: all random patterns gained fitness values between 0.25 and 0.30; all random sets of KIRLIAN images gained fitness values between 0.28 and 0.32; and all perceptual groups of KIRLIAN images gained fitness values from 0.34 to 0.53. Thus, the class of perceptual similar KIRLIAN images of the same subject is clearly distinguished from the class of random KIRLIAN images and random patterns, and this gives a justifiable evidence for intra-person specificity of KIRLIAN images.

4 Conclusions

In this paper, the fitness control procedure was introduced for the application of GA in image processing an pattern recognition tasks. Instead of deriving an optimization problem from the image processing problem and solving it by GA, a set of data bitstrings has to be derived from the problem, with similar data bitstrings representing the solution of a *selection* task. Then, the GA can be used for evolving a bitstring with high similarity to all data bitstrings, and the subgroup of most similar data bitstrings to that evolved bitstring (or the fitness value achieved) is taken as the result. Examples, where such a procedure might be applied, were given and discussed. The tasks were: main color selection, classification of text rows of a digitized invoice document into table rows and non-table rows; clustering of data by selecting the most suitable clustering algorithm among a set of concurrent clustering algorithms; and heuristic measuring of perceptual similarity of groups of images (KIRLIAN images in this case). The examples illustrate the pervasiveness of the proposed procedure, and demonstrates its general character.

References

[1] John H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.

[2] C. Bounsaythip and J. Alander. Genetic algorithms in image processing - a review. In *Proceedings of the Third Nordic Workshop on Genetic Algorithms and their Applications (3NWGA)*, pages 173–192, 1997.

[3] Mario Köppen, Dörte Waldöstl, and Bertram Nickolay. A system for the automated evaluation of invoices. In Jonathan H. Hull and Suzanne L. Taylor, editors, *Document Analysis Systems II*, pages 223–241. World Scientific, Singapore a.o., 1997.

[4] Hugues Juillé and Jordan P. Pollack. Co-evolving intertwined spirals. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Baeck, editors, *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*. A Bradford Book, MIT Press, Cambridge, MA, 1995.

[5] E. Dimitriadou W. Gablentz, M. Köppen. Robust clustering by evolutionary computation. In J. Tanskanen J. Martikainen, editor, *Proceedings WSC5 (CD-ROM)*, 2000.

[6] H. Treugut M. Köppen, B. Nickolay. Genetic algorithm based heuristic measure for pattern similarity in kirlian photographs. In E.J. Boers, editor, *Applications of Evolutionary Computing*, Springer Lecture Notes on Computer Science 2037, pages 317–324, 2001.