Rev.Acad.Canar.Cienc., V (Núm. 1), 95-110 (1993)

# Genetic algorithms applied to recognition problems

H. Van Hove
A. Verschoren *
University of Antwerp, RUCA
Department of Mathematics and Computer Science
Groenenborgerlaan 171
B-2020 Antwerp, Belgium

## 1. Introduction

Visual information usually contains a large amount of redundant data. For example, if we represent the numbers 0 to 9 by $6 \times 9$ bitmaps (printer characters, say), then we have, in principle, to consider 54 individual bits, in order to recognize a single character. In this way, we clearly perform too much work, since these 54 bits describe $2^{54} \sim 10^{16}$ possible bitmaps, whereas we are only interested in the much smaller set representing $\{0, \ldots, 9\}$. A better strategy thus consists in trying to discover significant bits that permit to differentiate between these bitmaps (*triggers*). The recognition procedure for a character will then consist in testing these particular bits. Checking whether a specific trigger is "on" or "off" may then lead, for example, to the conclusion that the character to recognize belongs to $\{0, 1, 3, 7\}$ or to $\{2, 4, 5, 6, 8, 9\}$, say. Within these classes other triggers may then be used in order to differentiate between 0 and 3 or 1 and 7, for example. It is clear that the recognition procedure for bitmaps may thus be modeled into the form of a binary tree, where each node will correspond to a test on a particular bit.

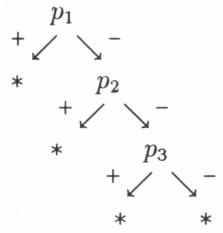
*This text is an expanded version of a talk given by the second author at the University of La Laguna. He wishes to thank the organizers for their warm hospitality.



© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

Somewhat more formally, let us start from a collection of objects $S$, the *problem space* (for example $1024 \times 1024$ bitmaps), endowed with a family of maps $f_i : S \to \{0,1\}$; the *feature maps* (for example, the individual bits for these bitmaps or a subset of these). Consider also a so-called *test space* $T \subseteq S$, whose objects we want to recognize (a particular sample of $1024 \times 1024$ bitmaps, for example). We want the objects in $T$ to be determined by the values of the triggers $f_i$, i.e, we want for any $t', t'' \in T$ that $f_i(t') = f_i(t'')$ for all $i$ implies $t' = t''$.

The *recognition problem* may then naively be formulated as: find a minimal collection of feature maps amongst the $f_i$, which still permits to differentiate between the objects in $T$. Of course, the term "minimal" is rather ambiguous here. Indeed, we may refer to the number of triggers needed or the number of tests to be actually performed, but also to the *cost* of performing these tests (calculation time or implementation difficulty). Moreover, the evaluations may be mutually dependent or may have to be performed in a specific order. It is thus clear that we should work directly with so-called *recognition trees*, which permit to recognize the objects in $T$. More precisely, we will be interested in binary trees of the form below, whose nodes correspond to certain triggers (or tests), whose edges are labeled by $\{+,-\}$ (expressing the outcome of the test corresponding to the node from where they start) and whose final nodes, labeled by $*$, should correspond bijectively to the objects to be recognized.




We assume to be given an externally defined *fitness function*, whose value expresses the overall value of the recognition tree with respect to the particular problem (e.g., the number of objects in $T$ which are actually recognized by the tree).

We will show in this text how a suitable version of genetic algorithms yields a solution to this problem, i.e., allows to construct recognition trees which behave optimally with respect to the objective fitness function. For a more



© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

detailed treatment and other applications, we refer to [9, 10, 11, 12] and work in progress.

## 2. Genetic Algorithms

**(2.1)** Genetic algorithms, introduced by J. Holland [5], are search algorithms based upon the mechanism of natural selection, as present in the general principles of Darwinism and genetics. In particular, they simulate ideas like "survival of the fittest", "mutation" and "structured information exchange". We refer to [4] for a survey of some of the domains, where they are succesfully applied.
The power of genetic algorithms resides in the following characteristics:

- genetic algorithms do not work directly upon the parameters of a problem, but on a suitable encoding of these;
- genetic algorithms do not work with a single object, but with a population of objects;
- genetic algorithms are "blind"; they do not work with special, additional information or characteristics, but only with direct information, given by the evaluation of an (objective!) fitness function;
- genetic algorithms do not work deterministically, but use randomized (probabilistic) transition rules.

To illustrate this, consider the "problem" of maximizing $f(x) = x^2$ on the interval $[0, 31]$. Instead of using traditional techniques (like hill-climbing and gradient methods), all of which are non-robust, one encodes the parameter space $[0, 31]$ by using 5-digit binary numbers and one works afterwards on the individual bits. One then starts from a random population of 5-digit binary strings, which is modified through the probabilistic genetic operators described below. Working with populations implies an implicit form of parallelism, which allows to find global extrema instead of local ones, thus eliminating one of the main pitfalls of classical algorithms.

**(2.2)** The simple genetic algorithm uses three elementary operators: *reproduction*, *crossover* and *mutation*. If we restrict ourselves to these operators,



© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

the algorithm proceeds as follows: Start from a (random) population $\mathcal{A}(0)$ of $N$ binary strings of length $\ell$, i.e., strings only consisting of 0 and 1. We denote the set of these (there are $2^\ell$ of them) by $\mathcal{R}_\ell$ . Each string $A_i \in \mathcal{A}(0)$ is given a fitness value $f_i$, determined by the fitness map $f : \mathcal{R}_\ell \to \mathbf{R}_+$, which describes the "quality" or "strength" of each string, depending upon the problem to be solved.

During the *reproduction phase*, the algorithm selects strings in $\mathcal{A}(0)$, where the probability of a particular string being selected is proportional to its fitness. Reproduction thus obviously mimicks the "survival of the fittest" principle.

To each pair of selected strings, one then applies *crossover* with a certain probability $p_c$, by choosing randomly a crossover site $1 \leq k \leq \ell - 1$, cutting both strings into two parts at this site and interchanging the pieces thus obtained. For example, if we start from the strings

$$A_1 : 1100 \mid 101$$
$$A_2 : 0110 \mid 110$$

where the crossover site ($k = 4$) is indicated by a vertical line ($|$), then we obtain, after crossover, two new strings

$$A_1' : 1100110$$
$$A_2' : 0110101$$

After this process, individual bits ("genes") are *mutated* ($0 \to 1, 1 \to 0$), with a very small probability, say $p_m = 0.001$. The latter operation is introduced in order to really induce changes, not inherently present within the components of the initial population and to reintroduce properties that got lost while applying the other operators.

We thus obtain a new population, denoted by $\mathcal{A}(1)$, on which the genetic algorithm may be applied again. For a precise description of the convergence features of this algorithm, we refer to [4, 5, et al].

**(2.3)** Given a population, *what* kind of information is contained within the strings in this population, that permits (through the fitness function) to apply directed search towards optimal values? If we consider the following example:



© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

| string | fitness |
|---|---|
| 01101 | 169 |
| 11000 | 576 |
| 01000 | 64 |
| 10011 | 361 |

(which corresponds to the example $f : \{0, \ldots, 31\} \to \mathbf{R} : x \mapsto x^2$, of course), then we obviously observe that strings starting with 1 have the highest fitness and that, of these, strings starting with 11 score better than those starting with 10. We thus see that individual strings possess *local* information (contained in the fitness of that string), while a population of strings possesses *global* information (the fitness of the composing strings, combined with structural information).

This structural, global information may be described through the use of schemata. A length $\ell$ *schema* (corresponding to a population of length $\ell$ strings) is a length $\ell$ string consisting of the symbols $0, 1$ and $\square$, where the latter should be viewed as a "wild card" (don't know, don't care) symbol. We say that a length $\ell$ string $A$ *satisfies* the schema $H$ (or that $H$ *represents* the string $A$), and we denote this by $H \to A$, if $A$ and $H$ coincide whenever possible, i.e., in these places where $H$ has a symbol different from $\square$.

For example, it is clear that

$$\square 1 \square 0 \to \begin{cases} 0100 \\ 0110 \\ 1100 \\ 1110 \end{cases}$$

If $H$ has *order* $o(H)$, i.e., if $H$ has exactly $o(H)$ bits different from $\square$, then there are exactly $2^{\ell - o(H)}$ strings $A$ with $H \to A$. Let us call *defining length* of a schema (denoted by $\delta(H)$) the distance between the first and the last significant bit of $H$. One may then show, cf. [4, 5], that schemata with high fitness (i.e., schemata which represent strings with fitness above average), low order and low defining length (which are usually referred to as *building blocks*) will receive an essentially exponentially increasing number of representatives throughout the successive generations produced by the genetic algorithm.

Without going into numerical or combinatorical details, let us just point out the following example. Consider the schemata



© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

| string | | | | | | $o(H)$ | $\delta(H)$ |
|---|---|---|---|---|---|---|---|
| $H_1$ : | 1 | □ | □ | □ | 0 | 2 | 4 |
| $H_2$ : | □ | □ | 1 | 1 | □ | 2 | 1 |
| $H_3$ : | □ | 1 | 0 | 1 | 0 | 4 | 3 |

Applying crossover on strings represented by these schemata, it is intuitively clear that those represented by $H_1$ and $H_3$ have highest probability of being destroyed (i.e., applying crossover to a string satisfying the schema will not necessarily yield a new string represented by it). The reason for this comes from the fact that $o(H_3)$ and $\delta(H_1)$ are "high". Somewhat more precisely, for any schema $H$, let us denote by $m(H,t)$ the number of representatives of $H$ within the population $\mathcal{A}(t)$. The influence of the genetic operators may then be described by

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\overline{f}}(1 - p_c\frac{\delta(H)}{\ell - 1} - o(H)p_m),$$

where $f(H)$ is the average fitness of the strings represented by $H$ and $\overline{f}$ is the average fitness of the whole population $\mathcal{A}(t)$. From this the above *Schema Theorem* follows easily.

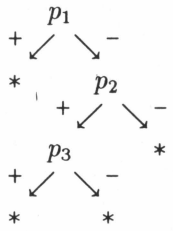
# 3. Recognition trees

**(3.1)** Consider a problem space $S$ endowed with a family of feature maps $f_1, \ldots, f_r : S \to \{0,1\}$. A *pattern* over $S$ is a string $p = a_1 \ldots a_r$, where $a_i \in \{0,1,\square\}$ should be viewed as the value (if determined) of the corresponding feature map $f_i$. We say that $s \in S$ *satisfies* the pattern $p = a_1 \ldots a_r$ if $f_i(s) = a_i$ for all $1 \leq i \leq r$ with $a_i \neq \square$. We write $\tau(p,s) = +$ or $\tau(p,s) = -$ depending on whether $s$ satisfies $p$ or not. A *recognition tree* of *width* $n$ is a binary tree with $n$ terminal nodes (labeled by $*$), whose other nodes are labeled by patterns and whose edge pairs are labeled by $\{+,-\}$. We denote the width of a recognition tree $A$ by $w(A)$.



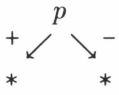
© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

**Example** ($n = 4$)

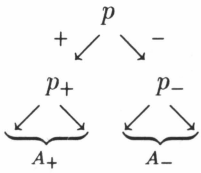



For each non-terminal node, labeled by the pattern $p$ say, the edge labeled by $+$ resp. $-$ starting in this node will be referred to as the *left* resp. *right edge* starting in $p$.

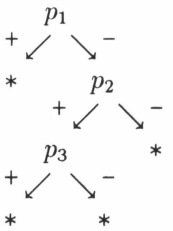
**(3.2)** Recognition trees may be encoded as follows. First, if $A$ is a 1-recognition tree, i.e., if $T$ has just one (final!) node, then we put $Enc(A) = *$. If $A$ is a 2-recognition tree, then it is of the form




and we encode it as $Enc(A) = (p * *)$. In general, if $A$ is an $n$-recognition tree with $n \geq 2$, say with top labeled by some pattern $p$, then the left and right edge starting in $p$ define "left" and "right" subtrees $A_+$ resp. $A_-$ of $A$. In a picture:




We then define (recursively) $Enc(A) = (p\ Enc(A_+)\ Enc(A_-))$.
For example, the 4-recognition tree $A$ given by

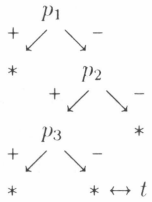





© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

is encoded as $Enc(A) = (p_1 * (p_2(p_3 * *) *))$. Each (non-terminal) node (labeled by the pattern) $p$ in a tree determines a subtree $A_p$ with top node $p$. We call $Enc(A_p)$ the *block* determined by $p$. For example, in the previous tree, $(p_2(p_3 * *) *)$ is the block determined by $p_2$.

**(3.3)** If $T \subseteq S$ in a test space, then we say that a recognition tree $A$ (with $w(A) = \mid T \mid$) *recognizes* $T$ if the objects $t \in T$ correspond bijectively to the final nodes of $A$ and if they realize each test in the unique path from the top of $A$ corresponding to the terminal node corresponding to $t$.
For example, if $t \in T$ should correspond to the terminal node as indicated in the figure below:




then we want:

$$\tau(p_1, t) = -$$
$$\tau(p_2, t) = +$$
$$\tau(p_3, t) = -$$

# 4. Genetic operators

**(4.1)** Let us start from a population $\mathcal{A}(0)$ of ($n$-)recognition trees and assume we are given a *fitness function* $f$, which to each recognition tree $A$ associates its fitness $f(A)$. This fitness function will usually consist of a *local* component (depending on the patterns occurring in the recognition tree $A$) and a *global* component (depending mainly on the structure of $A$ itself).
For example, if we want to find "optimal" recognition trees in the problem of recognizing a test space $T \subseteq S$ as in the previous section, then a typical fitness function could be given by letting

$$f(A) = \alpha_e E(A) + \alpha_r(\mid T \mid - r(A, T))$$



© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

for each recognition tree $A$. Here $E(A)$ denotes the average (or total) cost of evaluating the tests corresponding to the patterns in the tree, while $r(A,T)$ denotes the number of elements in $T$ recognized by $A$. The constants $\alpha_e$ and $\alpha_r$ are user defined parameters, which should be calibrated in terms of the user's desiderata (what do we prefer: recognizing a maximal number of objects in $T$ or minimizing the global cost of the recognition process?).
Let us now introduce, as in [10], operators to be used by the genetic algorithm.

**(4.2) Reproduction.** The reproduction operator in $\mathcal{A}(0)$ and its successive generations works in exactly the same way as for the ordinary genetic algorithm. So, recognition trees will be selected with a probability proportional with their fitness. If we denote this probability by $p_A$, then

$$p_A = \frac{f(A)}{\sum_{B \in \mathcal{A}(0)} f(B)}.$$

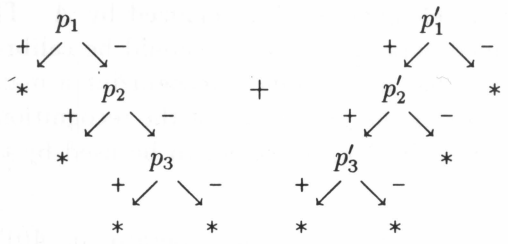
Again, the argument for this resides in the fact that recognition trees with high fitness are likely to produce offspring of at least equal (and possibly higher) fitness. This selection procedure is usually implemented by using a "roulette model", i.e., one views the recognition trees as being distributed over a roulette, each one occupying a sector of surface proportional to its fitness.

**(4.3) Crossover.** The crossover operator is the analogue for recognition trees of the usual crossover for strings. For any node (labeled by a pattern) $p$ in a recognition tree $A$, one denotes by $n_A(p)$ the width of the subtree of $A$ with top $p$. If two recognition trees $A$ and $A'$ are selected (through reproduction, for example), then one selects in $A$ randomly a non-terminal node. If $n_{A'}(p') \neq n_A(p)$ for all nodes $p'$ in $A'$, crossover does nothing. Otherwise, it selects $p'$ randomly with $n_A(p') = n_A(p)$. One then interchanges in $A$ and $A'$ the subtrees $A_p$ and $A'_{p'}$ with top $p$ resp. $p'$ and thus obtains two new recognition trees $B$ and $B'$.

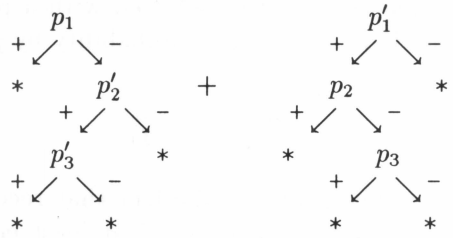


© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

So, in a picture, starting from:




yields



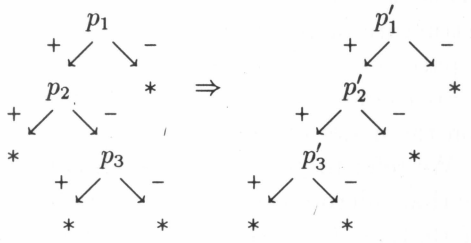

The encoding $Enc(B)$ of $B$ is found from that of $A$ by exchanging the block defined by $p$ in $Enc(A)$ by the block defined by $p'$ in $Enc(A')$.
**Example**: If $Enc(A) = (p_1 * (p_2 * (p_3 * *)))$ and $Enc(A') = (p'_1(p'_2(p'_3 * *)*)*)$, then (if crossover is performed with respect to $p_2$ and $p'_2$) we obtain new recognition trees $B$ and $B'$ with $Enc(B) = (p_1 * (p'_2(p'_3 * *)*))$ and $Enc(B') = (p'_1(p_2 * (p_3 * *))*)$.

**(4.4) Switch.** The switch operator for recognition trees may be viewed as a global analogue of the mutation operator on strings. It depends on the choice of a node $p$ in a recognition tree $A$. If $p$ is a terminal node, the switch operator does nothing. Otherwise, it interchanges the left and right edges starting in $p$, in order to produce a new recogniton tree $B$.



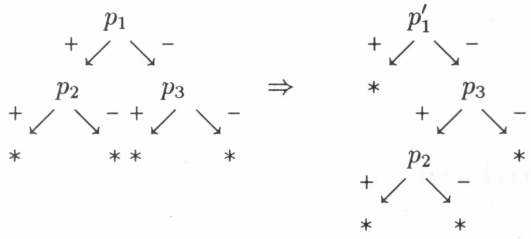
© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

In a picture:




The encoding $Enc(B)$ of $B$ may be found from that of $A$ by interchanging in $Enc(A)$ the second and third component in the block determined by $p$.
**Example** : If $A$ is encoded by $Enc(A) = (p_1(p_2 * (p_3 * *))*)$ then (if switch is applied in $p_2$), we find for $B$ the encoding $Enc(B) = (p_1(p_2(p_3 * *)*)*)$.

**(4.5) Translocation.** The translocation operator for recognition trees may be viewed a an "auto-crossover" operator; it should be compared to linear inversion [3] and translocation [8] for the ordinary genetic algorithm on strings. It depends on the choice of a node $p$ in a recognition tree $A$. If $p$ is a terminal node, then the translocation operator does nothing. Otherwise, it chooses randomly a terminal node in $A$ which does not belong to the subtree $A_p$ of $A$ with top $p$ and it interchanges this terminal node and $A_p$, and thus producing a new recognition tree $B$.
In a picture:




The encoding $Enc(B)$ of $B$ may be found from that of $A$ by interchanging in $Enc(A)$ some $*$ and the block determined by $p$.
**Example** : If $Enc(A) = (p_1(p_2 * *)(p_3 * *))$ and if we apply translocation as in the previous example, then we obtain a new recognition tree $B$ with $Enc(B) = (p_1 * (p_3(p_2 * *)*))$.



© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

**(4.6) Micro-operators.** The main purpose of the previous operators is to modify the global structure of recognition trees, but they do not influence their contents (the patterns which label their nodes). In order to remedy this, one introduces micro-operators like micro-crossover and micro-mutation which act directly on the nodes of the recognition trees in $\mathcal{A}(0)$ and its successive generations. We refer to [9, 10] for a detailed treatment of these. Let us just mention here that micro-crossover is essentially ordinary crossover for strings composed of $\{0, 1, \square\}$ (= patterns) occurring at nodes of recognition trees with high fitness. Micro-mutation works in a similar way.
It may be proved that the relevance of these operators is higher for recognition trees with low width. In the general case, they should be applied with bigger probability in the *beginning* of the algorithm, relaxing this frequency gradually afterwards (cf. simulated annealing).

**(4.7) The algorithm.**

- Start from a random population of $N$ ($n$-)recognition trees.
- Select, through reproduction, two recognition trees and apply
  - crossover with probability $p_c$;
  - switch with probability $p_s$;
  - translocation with probability $p_t$;
  - micro-operators with probability $p_{\mu c}$ and $p_{\mu m}$.
- Iterate (until a suitable population $\mathcal{A}(t)$ is obtained).

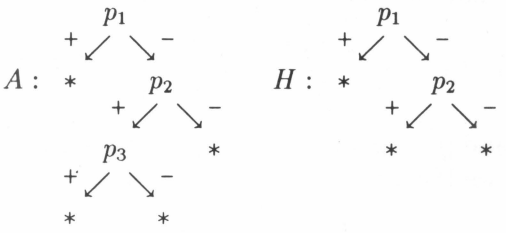
# 5. Recognition schemata

**(5.1)** Just as in the classical situation, where schemata are used to describe populations of strings, the global structure of a population of $n$-recognition trees may be studied by using socalled $n$-recognition schemata. Here, an $n$-recognition schema is just an $r$-recognition tree $H$ with $r \leq n$. We say that $H$ *represents* an $n$-recognition tree $A$ (or that $A$ *satisfies* $H$) and we denote this by $H \rightarrow A$, if there exists an embedding of $H$ into $A$. This means that $H$ may be recovered as a subtree of $A$, up to replacing some non-terminal nodes in $A$ by $*$.



© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

For example with $A$ and $H$ as below we have $H \rightarrow A$:




If $H \rightarrow A$, then it is also clear that

$$Enc(A) = (\ldots Enc_0(H) \ldots),$$

where $Enc_0(H)$ is obtained from $Enc(H)$ by (possibly) replacing some terminal symbols $*$ by patterns.

**(5.2)** If "most" representatives of some fixed recognition schema $H$ by recognition trees in a population $\mathcal{A}(0)$ have high fitness, then this seems to imply that the presence of the "pattern" $H$ is the origin of the high fitness of these representatives. More evidence for this arises if $H$ tends to "survive" through the successive generations generated by applying the genetic algorithm. So, for each positive integer $t$, let $\mathcal{A}(H,t)$ be the set of all recognition trees $A \in \mathcal{A}(t)$ with the property that $H \rightarrow A$ and let us write $m(H,t) = \mid \mathcal{A}(H,t) \mid$. The efficiency of the genetic algorithm and its effect on $H$ may be understood by considering the value of $m(H,t)$ through successive generations.

**(5.2.1)** If only reproduction is applied, then, exactly as is the classical case, we obtain

$$m(H,t+1) = m(H,t)\frac{f(H)}{\overline{f}}.$$

It is thus clear that schemata with high fitnes will receive an increasing number of representatives. If we assume, as a first approximation, that $f(H)$ constantly remains above the average, say $f(H) = \alpha f$ with $\alpha > 1$, then we obtain $m(H,t+1) = \alpha m(H,t)$, so $m(H,t) \sim \alpha^t m(H,0)$. The number of recognition trees in $\mathcal{A}(H,t)$ thus increases exponentially, in this case. One may show similarly that if $H$ scores below average, then $m(H,t)$ decreases exponentially.



© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

**(5.2.2)** If the other operators are also applied, then some straightforward combinatorial calculations (see [9, 10] for full details) show that

$$m(H,t+1) \geq m(H,t)(1-p_c\frac{r-2}{n-2})(1-p_s\frac{r-1}{n-1})(1-p_t\frac{r-2}{n-2})(1-p_\mu\frac{r-1}{n-1}),$$

where $r = w(H)$ and $p_\mu = p_{\mu c} + (n-1)p_{\mu m}$. If we assume the above probabilities to be low, we then, by omitting crossed products, get

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}(1-(p_c+p_t)\frac{r-2}{n-2}-(p_s+p_\mu)\frac{r-1}{n-1}).$$

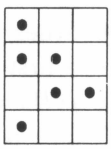
We thus obtain

**(5.3) Theorem.** *(Schema Theorem) The genetic algorithm attributes to recognition trees with low width and high fitness an exponentially increasing number of representatives through the successive generations.*

# 6. Concluding remarks

**(6.1)** The previous techniques may also be applied to strategy trees. These only differ from recognition trees, in the fact that they do not necessarily have a fixed number of terminal nodes. They obviously find their origin within game theory, where they are used to model 2-player strategy games. The results obtained in [7] show that genetic algorithms may be used on these trees, to find optimal strategies, for example.

**(6.2)** In principle, visual information (bitmaps) may be encoded linearly. For example, the $4 \times 3$ bitmap



© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

may be encoded as 100 110 011 100 (using rows) or as 110 101 100 010 (using columns). Although this works perfectly well if we want to apply the genetic algorithm, we may increase its speed and efficiency by working directly with a two-dimensional matrix instead of with linear encodings. Introducing suitable genetic operators for these "two-dimensional strings", one may thus take into account two-dimensional correlations (hardly visible in linear encodings) and obtain the desired enhancement of the genetic algorithm. We refer to [9, 10] for details.

# References


[1] Bagley, J.D., The behaviour of adaptive systems which employ genetic and correlation algorithms, ph.d. thesis, University of Michigan, 1967.

[2] Cavicchio, D.J., Adaptive search using simulated evolution, ph.d. thesis, University of Michigan, Ann Arbor, 1970.

[3] Frantz, D.R., Non-linearities in genetic adaptive search, ph.d. thesis, University of Michigan, Ann Arbor, 1972.

[4] Goldberg, D.E., Genetic algorithms in search, optimization and machine learning, Addison-Wesley, 1989.

[5] Holland, J.H., Adaptation in natural and artificial systems, University of Michigan Press, 1975.

[6] Krishnaiah, P.R. and L.N. Kamal, Classification, pattern recognition and reduction of dimensionality, North Holland, 1982.

[7] Robeys, K., H. Van Hove and A. Verschoren, *Genetic algorithms and trees, part II: Strategy trees*, Computers and Artificial Intelligence, submitted.

[8] Smith, S.F., A learning system based on genetic adaptive algorithms, ph.d. thesis, University of Pittsburgh, 1980.

[9] Van Hove, H., Twodimensional genetic algorithms, ph.d. thesis, University of Antwerp, in preparation.


© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017

[10] Van Hove, H. and A. Verschoren, *Genetic algorithms and trees, part I: Recognition trees*, Computers and Artificial Intelligence, submitted.

[11] Van Hove, H. and A.Verschoren, Two-dimensional genetic algorithms, to appear.

[12] Van Hove, H. and A. Verschoren, Genetic algorithms applied to recognition problems, University of Antwerp, RUCA, preprint, 1992.

Recibido: 30 de Diciembre 1993

© Del documento, de los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2017