STUDIA UNIV. "BABEŞ-BOLYAI", INFORMATICA, Volume XLIII, Number 1, 1998

EVOLUTIONARY PROGRAMMING: AN APPLICATION TO CLUSTERING

DANA AVRAM DAN DUMITRESCU
BEATRICE LAZZERINI

Abstract. In this paper, we present the two basic models of Evolutionary Programming. The first model tries to obtain intelligent predictions by using a determinist finite-state automation or a sequential machine. The second model refers to the optimization of real functions. This second model is exemplified by an application of clustering.

1. Introduction

Evolutionary Programming (EP) represents a class of paradigms for simulating evolution which utilizes the concepts of *Darwinian evolution* to iteratively generate increasing appropriate solutions (organisms) in light of a static or dynamically changing environment. This is in sharp contrast to earlier research into artificial intelligence research, which largely centered on the search of simple heuristics. Instead of developing a (potentially) complex set of rules, which were derived from human experts, EP evolves a set of solutions, which exhibit optimal behavior with respect to an environment and a desired payoff function.

Evolutionary Programming (Fogel, 1962, Fogel, Owens, Walsh, 1966) appeared as the result of the effort to generate the intelligent behavior of a machine described by a determinist finite-state automaton. In this case, the intelligent behavior is a prediction of the environment described as a succession of input symbols of the automaton.

In order to find a machine capable to offer a better prediction, we will consider a machines randomly generated population of machines. The different parts of the machines (states, transitions, output symbols) are modified later. The machines are compared by means of an evaluation function. The most productive states are chosen. The choice is strictly deterministic. The recombination of two automata does not seem to be an interesting idea. That is why the only search operator is mutation.

The ideas used in Evolutionary Programming surpassed the initial domain and were also applied to solve the optimization and search problems (D. B. Fogel, 1992). The EP optimization algorithm has many things in common with the evolution strategies.

In what follows we present the basic elements of Evolutionary Programming. The two basic models of EP are described. The first model tries to obtain intelligent

---

1991 *Mathematics Subject Classification.* 68H30, 68T99.
1991 *CR Categories and Subject Descriptors.* I.5.3 [**Pattern Recognition**]: Clustering - *algorithms, similarity measures.*

predictions by using a determinist finite-state automaton or a sequential machine. The second model refers to the optimization of real functions.

The operators of Evolutionary Programming are selection and mutation. The selection is of type $(\mu+\mu)$ and it is deterministic. The mutation is realized by a normal random perturbation. The standard deviation σ of the normal distribution is a parameter of the method. An self-adaptation mechanism controls the evolution of this parameter.

There are different methods to control the parameters. One of them is known as *Meta Evolutionary Programming*.

2. Finite Automaton

In this section we present some notions concerning the finite-state automaton.

An automaton is an abstract machine having a finite set of *states*. An input tape contains a set of *symbols*. *The input symbols* are read sequentially. An input symbol and a state determine the transition of the automaton to another state.

In order to give a formal definition of automaton, we will introduce the following notations.

(i) Σ is the set of input symbols. Σ defines the *input alphabet* of the automaton.

(ii) $\mathbf{S}$ is a finite and not empty set called *the set of the states* of the automaton.

(iii) $\mathbf{S_0} \in \mathbf{S}$ is *the initial state* of the automaton.

(iv) The function $\delta : \mathbf{S} \times \Sigma \to \mathbf{S}$ is *the transition function* of the automaton.

A *determinist finite-state automaton* is a system $\mathbf{A} = (\Sigma, \mathbf{S}, \mathbf{S_0}, \delta)$, where the signification of the symbols is the one given before.

The transition function describes the dynamic of the automaton. The equality

$$s' = \delta(s, x) \quad , \quad (s, x) \in S \times \sum \quad ,$$

indicates that the automaton is in the state **s**, the input symbol **x** and passes into the state **s'**.

An input symbol corresponds to a certain situation of the environment where the automaton is placed. By reading an input sequence (representing the action of the environment) the automaton passes from the initial state $\mathbf{S_0}$ into another state.

We can complicate the model of the finite-state automaton by introducing an *output tape* where an output symbol is recorded. In this way we obtain a sequential machine.

A *sequential machine* (or an input-output automaton) is a system

$$\mathbf{T} = (\Sigma, \mathbf{S}, \mathbf{S_0}, \mathbf{O}, \delta)$$

where the symbols Σ, $\mathbf{S}$ and $\mathbf{S_0}$ have the signification indicated above. $\mathbf{O}$ is *the set of output symbols* and the transition function δ is a function:

$$\delta : \mathbf{S} \times \Sigma \to \mathbf{S} \times \mathbf{O} \quad .$$

The equation

$$(s', o) = \delta(s, x)$$

is interpreted as follows: the sequential machine in the state **s** reads an input symbol **x** passes into another state **s'**, produces and writes on the output tape the output symbol **o**.

Using this mechanism, the sequential machine transforms a sequence of input symbols into a sequence of output symbols. A sequential machine can simulate a kind of

intelligent behavior. In this case the intelligence can be defined as the ability of the system to adapt its behavior so that it can realize the desired outputs in various extreme situations.

3. Evolutionary Programming. Sequential Machine Model

Initially, the main goal of Evolutionary Programming was to operate with sequential machine and with its discrete representations. The intention was to obtain predictions of the environment. In this case, the intelligent behavior was seen as the ability of machine to obtain reasonable predictions of its environment. Those predictions are translated into responses regarding a domain. The environment is described as a sequence of input symbols. The prediction is represented by the output symbol. The performance of the machine is measured using an evaluation function (or error function).

The machine works as follows. An output symbol represents a prediction of the next state of the environment. Thus the quality of the prediction can be evaluated by comparing the output symbol with the next input symbol. The distance between the output symbol and the next input symbol is the *error* of the prediction. The prediction quality is measured by using an evaluation function.

The Evolutionary Programming paradigm uses a population of $\mu > 1$ parents. Each parent represents a sequential machine. From μ parents the mutation operator generates μ offspring. The mutation represents random changes of the elements composing each parent machine. There are five possible ways to perform the mutation:

(i) Change an output symbol.
(ii) Change a state transition.
(iii) Add a new state.
(iv) Delete an existing state.
(v) Change the initial state.

Obviously, to delete a state or to change the initial state is possible only if the machine has more than one state.

The number of the mutation operations applied to each offspring is also determined with respect to a probability distribution or may be fixed *a priori*. Usually, this probability distribution is the uniform distribution.

The μ offspring obtained by using mutation are evaluated with respect to the environment. The evaluation uses a fitness function. μ individuals are chosen among the parents and the offspring. They will be the parents of the new generation. The selection is made in the decreasing order of the quality. The used selection is of the type $(\mu+\mu)$ and is deterministic.

The process continues until a machine achieving a correct prediction of the next symbol (not yet explored of the environment) appears in a population. The machine that accomplishes the correct prediction is selected to generate that prediction. The new symbol is added to the already explored environment and the process continues.

Basically, this mechanism tries to obtain machines that modify their behavior in order to increase their capacity to predict correctly a set of symbols describing the environment. This is a characteristic of a learning process. The symbols describing the environment can be seen as a training set.

We can summarize the discussion above into an Evolutionary Programming algorithm for evolving sequential machines.

Evolutionary Programming Algorithm for Sequential Machines

S1. Set **t** := 0 ;
Initialize by random a population **P(t)** of μ parents. Each member of the population is a finite-state sequential machine.
Suppose **L** is the set of input symbols that has already been checked.
Set L := ∅ .

S2. The environment (training set) is presented to the members of the parent population **P(t)**. The training set consists of input symbols.
Each input symbol is presented to each machine of population **P(t)**.
The output symbol of each machine (the prediction) is compared to the next input symbol.

S3. The quality of the prediction is calculated for each input symbol and for each machine by using a fitness function.

S4. The population **P(t)** is evaluated.
To reach this goal, each machine fitness for an input sequence is calculated. The fitness of a machine can be defined, for example, as the average performance (with respect to the input symbols).

S5. μ machines are created (μ offspring) by applying mutation to μ parents of the population **P(t)**. Denote by **P'** the population of those new machines.

S6. The machines of the population **P'** are evaluated according to the existing environment in the same way as their parents were.

S7. μ individuals are chosen from the population **P(t)** ∪ **P'**. The choice is done by selecting μ individuals in the increasing order of their fitness.
The selected individuals will be the parents of the new generation.
Set t := t + 1 .

S8. The steps S5-S7 are repeated until a machine achieves a correct prediction of an input symbol not yet checked.
This machine is selected to generate this prediction.
The symbol that was correctly predicted is added to the list **L** of checked symbols.
The process returns to the step S2.

4. Evolutionary Programming Used for Function Optimization

Fogel (1992) extended the Evolutionary Programming to optimize the real functions. The representation of individuals, the mutation operator and self-adaptation of strategy parameters have many common aspects with the corresponding elements in Evolution Strategies. The Evolutionary Programming is not supposed to use a recombination operator in this case either.

4.1. Chromosomes Representation

In Evolutionary Programming, a chromosome corresponds to an object variables vector. Consider the objective function $\mathbf{F} : \mathbf{R}^n \rightarrow \mathbf{R}$ and no constraint optimization problem:

$$\begin{cases} F(x) \longrightarrow \max \\ x \in \boldsymbol{R}^n \end{cases}$$

In this case, the chromosome **x** is a vector from searching space $\mathbf{R}^n$.

The *Meta Evolutionary Programming* (Fogel, 1992) represents individuals in a slightly different way. The Meta Evolutionary Programming includes a self-adaptation parameter mechanism that is similar to the one used in Evolution strategies. The control parameter of the method is the standard deviation (or the square of standard deviation that is dispersion).

In Meta Evolutionary Programming an individual **a** is a pair $\mathbf{a}=(\mathbf{x},\sigma^2)$, where σ^2 is a vector. The components of this vector are the dispersion components:

$$\sigma^2 = \begin{pmatrix} \sigma_1^2 \\ \sigma_2^2 \\ \dots \\ \sigma_n^2 \end{pmatrix}$$

The fitness function is obtained starting from the objective function by scaling it to positive values. It is possible to introduce some random alterations inside the objective function.

Consider **S** a set of additional parameters involved in the process. Let us admit that these parameters can be randomly modified. The scaling function is:

$$\alpha : \mathbf{R} \times \mathbf{S} \rightarrow \mathbf{R}^+ \quad .$$

The fitness function **f** can be written as follows:

$$f(a) = \alpha(F(x), k) \quad ,$$

where **k** describes the random changes of the process parameters.

4.2. Mutation

In Evolutionary Programming, as in the Evolution Strategies, the mutation of the components of the vector **x** is obtained by using a random perturbation. The perturbation of the components is additive. Thus we will have:

$$x_i' = x_i + \sigma_i N_i(0,1) \quad , \quad i = 1, 2, \dots, n$$

where $\mathbf{N_i(0,1)}$ indicates a realization of a normal random variable with standard deviation one and expectation zero.

The dispersion is considered to be proportional with the square root of the value corresponding to the fitness function:

$$\sigma_i^2 = \beta_i f(x) + \gamma_i$$

where the parameters β_i and γ_i are chosen according to the concrete problem.

Often the next values are used:

$$\beta_i = 1, \quad \gamma_i = 0,$$

In this case the components change according to the rule

$$x_i' = x_i + N_i(0,1)\sqrt{f(x)}$$

In order to avoid the difficulties regarding the adjustment of the parameter a *self adaptation* mechanism for the dispersion parameter has been created. This mechanism is similar to the Evolution Strategy mechanism.

For the first time, an additive variant of the self adaptation process has been considered:

$$\sigma_i' = \sigma_i(1 + \alpha\, N(0,1)) \quad ,$$

where α is a positive real parameter, with a subunitary value ($\alpha \approx 0.2$). This adjusting rule has proved to be inefficient for fitness functions altered by some distortions (noises). In those cases it can happen that the additive mechanism for self adaptation reverses the search direction.

The *Meta Evolutionary Programming* is a slightly modified variant of the self adaptation mechanism described above. Independent parameters which are self-adapting by using normal perturbations are considered. In this case, the mutations of the object variables and parameters are made according to the next rules:

$$x_i' = x_i + \sigma_i\, N_i(0,1), \qquad D_i' = D_i + \sigma_i \sqrt{\alpha}\, N_i(0,1),$$

where Di denote

Here α designates an external parameter that ensures that $\mathbf{D_i}$ tends to remain positive. If the variance $\mathbf{D_i}$ becomes negative or zero, it is modified into a small positive value, $\varepsilon > 0$.

The parameter self adaptation rule can be written as follows

$$\sigma_i'^2 = \sigma_i^2 + \sigma_i \sqrt{\alpha}\; N_i(0,1) \quad .$$

We remark that the two self adaptation rules (for variables and parameters) are quite similar.

It is interesting to compare additive rules of self adaptation and the multiplicative rule used in Evolution Strategies. Here there are some conclusions resulted from this comparison:

(i) The multiplicative self adaptation rule in case of Evolution Strategies ensures the positiveness of the strategy parameters;

(ii) The perturbations generated by the additive rule of the Meta Evolutionary Programming are greater then the perturbations of the exponential multiplicative law of the Evolution Strategies;

(iii) In the absence of the selection pressure, the perturbations of a strategy parameter are neutral. The additive rule does not generate neutral perturbations.

(iv) Numeric experiments indicate that the multiplicative self adaptation method seems to be more robust than the additive method. The robustness exists also because the transition between small and great values of control parameters in Evolution Strategies is easier to be accomplished.

Due to the advantages of the multiplicative rule we can also use this rule for Evolutionary Programming. The self adaptation multiplicative rule is often used in Evolutionary Programming implementations.

4.3. Selection

Let us consider an initial population of μ individuals (**t=0**). This will be the parent population for the next generation. Let **P(t)** be the parent population. Using the mutation only once for each parent we obtain μ offspring. Let **P'** be the offspring population. From the μ parents and μ offspring we select μ individuals. The selection method is a probabilistic variant of the selection $(\mu+\mu)$.

Each individual $\mathbf{a}^i$ of the population $\mathbf{P(t)} \cup \mathbf{P'}$ is compared to $\mathbf{q}>1$ individuals randomly chosen from $\mathbf{P(t)} \cup \mathbf{P'}$. The comparison is made by using the fitness function. For each individual $\mathbf{a}^i \in \mathbf{P(t)} \cup \mathbf{P'}$ the number of individual less fitted than $\mathbf{a}^i$ is calculated. This number represents the score $\mathbf{w}_i$ of $\mathbf{a}^i$ ($\mathbf{w}_i \in \{ 0, 1, \ldots, q \}$). After the score calculation, the 2μ individuals are sorted in the increasing order of the scores, $\mathbf{w}_i$, $\mathbf{i} = 1, 2, \ldots, 2\mu$. The μ individuals that have the best scores are selected. The selected individuals will form the new population **P(t+1)**.

The score $\mathbf{w}_i$ can be written as follows

$$w_i = \sum_{j=1}^{q} \begin{cases} 1 , & \text{if } f(a^i) \le f(a^{k_j}) \\ 0 , & \text{otherwise} \end{cases}$$

where the indexes $\mathbf{k}_j \in \{ 1, 2, \ldots, 2r \}$ are the values of a random uniform variable. $\mathbf{k}_j$ is recalculated for each comparison.

The selection obtained is a variant of probabilistic selection of the type q-tournament selection. **q** represents the number of the individuals in competition and it is a parameter of the method. Usually it is considered **q=100**. While the values of **q** are increasing the selection mechanism becomes closer to the deterministic schemata **(r+r)**.

For greater values of the **q** parameter, the selection schemata becomes *elitist*. In this situation the probability that the best individual gain the maximum score increases.

4.4. Recombination

Evolutionary Programming does not use a recombining operator. In the Evolutionary Programming a solution encodes a species not an individual. At the biological level the crossover does not function among different species. The biological model used does not give a conceptual difference between Evolution Strategies and Evolutionary Programming.

4.5. Evolutionary Programming Algorithm

The algorithm of Meta Evolutionary Programming used to find the optimum of a real function can be described by using the next standard form:

Evolutionary Programming Algorithm for the Real Function Optimization

S1. Set **t := 0** ;
Initialize a population **P(0)** of potential solutions.
S2. Evaluate the population **P(t)**.
S3. While **(T(P(t)) = false) execute** {the stop condition is not fulfilled }

P'(t) := mutation within **P(t)** ;
Evaluate **P'(t)** ;
P(t+1) := selection within **P(t)** $\cup$ **P'(t)** .
t := **t** + **1** .

4.6. Convergence of the Method

Fogel (1992) analyzed the convergence of the standard Evolutionary Programming algorithm when the fitness function is the objective function.

Consider a particular case when the discrete search space is $\mathbf{C}^n \subset \mathbf{R}^n$, where **C** is the set of numbers that can be represented into a numeric computer. In this case the probability that the algorithm globally converge is equal with 1.

Because the similarity between the (1+1) Evolution Strategy and Evolutionary Programming we can suppose that the convergence theorem for the (1+1) Evolution Strategy can be extended also for the Evolutionary Programming. We must remember that the convergence is probabilistic also in the case of Evolution Strategy.

Thus, the (1+1) convergence theorem can be extended also for the standard Evolutionary Programming. The convergence is true for the searching space $\mathbf{R}^n$ and not only for the discrete space considered above.

For the simplified sphere model with the fitness function:

$$F(x) = \sum_{i=1}^{n} (x_i)^2 = r^2,$$

the theory of the convergence rate of the (1+1) Evolution Strategy can be extended for the Evolutionary Programming. In this last situation for the case of a population with the size μ=**1**, the Evolutionary Programming selection becomes deterministic.

Consider the case when the dispersion has the expression:

$$\sigma_i^2 = \beta_i f(x) + \gamma_i \quad .$$

The convergence rate decreases to zero when and **n** increases. But if

$$\beta_i = \frac{1}{n^2} \quad ,$$

the convergence rate becomes almost optimal.

4.7. Conclusion

This Evolutionary Programming is basicaly similar with the Evolution Strategies. Although there are some differences between the two approaches. Two of them seem to be essential.

(i) The Evolution Strategy codifies structure that are similar to the individuals. This is why we can use recombination to obtain new individuals. Usually, the Evolutionary Programming that are similar to different species (and therefore the recombination operator cannot be applied for such structures). A mechanism to obtain new solutions by using recombination is not appropriate for the Evolutionary Programming.

(ii) Evolution Strategies use a strictly deterministic selection (the best individuals chosen among the parents and their offspring form the new generation).

The Evolutionary Programming uses a probabilistic selection. Each solution is compared to a fixed number of solutions randomly chosen from the current generation and not to all the other solutions from the current generation.

The Evolutionary Programming benefits from some ideas and concepts emerged within the Evolution Strategies.

The Evolutionary Programming suggests a possible amelioration for the Evolution Strategies. The probabilistic mechanism of selection based on sorting could be used in Evolution Strategies (μ,λ) and $(\mu+\lambda)$. Obviously there are also other selection mechanisms that could prove themselves useful. But there is no reason for using other selection mechanisms for the Evolutionary Programming or for the Evolution Strategies.

There exists a strong relationship between Evolutionary Computation and some other techniques, e.g. fuzzy logic and neural networks, usually regarded as elements of artificial intelligence. Their main common characteristic lies in their numerical knowledge representation which differentiates them from traditional symbolic Artificial Intelligence. Bezdek suggested the term *Computational Intelligence* for this special branch of Artificial Intelligence with the following characteristics:

1. numerical knowledge representation;
2. adaptability;
3. fault tolerance;
4. processing speed comparable to human cognition processes;
5. error rate optimality (estimate of the probability of a certain error on future data).

The Evolutionary Programming strategy applications are numerous. Part of them are related to Artificial Intelligence. Those are: neural networks training and design, pattern recognition, robotics, automated learning processes and searching in the state space, the system identification and system control.

The current direction for research regards some basic mathematical aspects, the combination of Evolutionary Programming with other searching techniques and the algorithm design for implementing them on parallel machines.

Other important research is conducted into the understanding of the *convergence properties* of EP, as well as the mechanisms of different mutation operators and selection mechanisms. The number of application areas of this optimization technique is constantly growing. EP, along with the other EC techniques, is being used on previously untenable problems which occur quite often in commercial and military problems.

5. Evolutionary Programming Based Clustering

Let $\mathbf{X}=\{\mathbf{x}^1, \ldots, \mathbf{x}^k\}$, $\mathbf{x}^j \in \mathbf{S}$ be a data set.

The cluster structure of $\mathbf{X}$ is described by a fuzzy partition $\mathbf{P}=\{\mathbf{A}_1, \ldots, \mathbf{A}_n\}$ of $\mathbf{S}$. $\mathbf{L}^i \in \mathbf{S}$ denotes the prototype of the fuzzy class $\mathbf{A}_i$.

The inadequacy $\mathbf{J(P,L)}$ of representing the fuzzy partition $\mathbf{P}$ by $\mathbf{L}=\{\mathbf{L}^1, \ldots, \mathbf{L}^n\}$ may be defined as:

$$J(P,L)=\sum_{i=1}^{n}\sum_{j=1}^{p} A_i^m\left(x^j\right)d^q\left(x^j,L^i\right) \quad ,$$

where **m**≥1 and **q**≥1.
Solving the optimization problem

$$J(P,L) \rightarrow min \, ,$$

is not an easily task for an arbitrary distance function d or for an arbitrary data space **S**.

Evolutionary Programming may offer a general framework for solving clustering problem.

5.1. Chromosome Representation

Each chromosome **c** is a prototype sequence

$$c = (L^1, \ldots, L^n) \, .$$

Real representation seems to be more appropriate for clustering problems than binary representation.

5.2. Fitness Function

The fitness function may be defined as:

$$f(c) = K_t - J(P,c) \, ,$$

where P is the partition induced by the chromosome $\mathbf{c} = (\mathbf{L^1}, \ldots, \mathbf{L^n})$. $\mathbf{K_t}$ is the maximum objective function value for the generation **P(t)**:

$$K_t = \max_{c \in P(t)} J(P,c)$$

For **q**=2 we may consider the fuzzy partition defined as:

$$A_i\left(x^j\right)=\frac{1}{\sum_{k=1}^{n}\left(\frac{d^2\left(x^j,L^i\right)}{d^2\left(x^j,L^k\right)+\alpha}\right)^{\frac{1}{m-1}}+\alpha} \quad , \qquad i=1,\ldots,n; \quad j=1,\ldots,p, \quad \alpha>0.$$

We may also obtain a hard partition **P** by using the 1- prototype rule:

$$\mathbf{x} \text{ is assigned to } \mathbf{A_i} \;\Leftrightarrow\; d\left(x,L^i\right)=\max_j d\left(x,L^j\right) \, .$$

5.3. Mutation and Self Adaptation

Let $\mathbf{c_i}$ be the **i**-th gene of the chromosome **c**. Consider each gene will be mutated with a standard deviation σ_i. The self adaptation rule of the dispersion is:

$$\sigma_i'^2 = \sigma_i^2 + \sigma_i \sqrt{\alpha} \; N_i(0,1)$$

when $\alpha \in (0,1)$.

The genes (object variables) are mutated according the additive rule:

$$c_i' = c_i + \sigma_i' \, N_i(0,1)$$

5.4. Survival

We use general EP survival mechanism.

5.5. Numerical examples

In order to get a good visual display of our genetic algorithm, we choose the simple example of clustering two-dimensional data into four clusters. Our example considered 75 generations and the points from [0,1] interval. Notice that the prototypes are marked with an "x" and data points are marked with an "o".

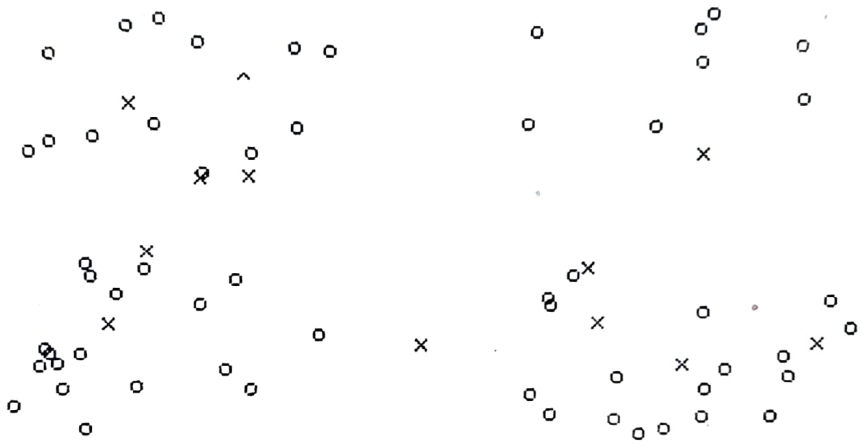

Figure 1. Generation 0.

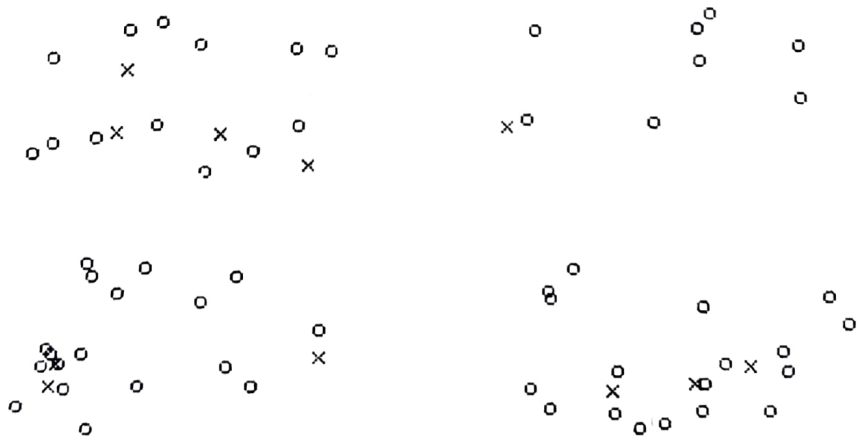

Figure 2. Generation 15.

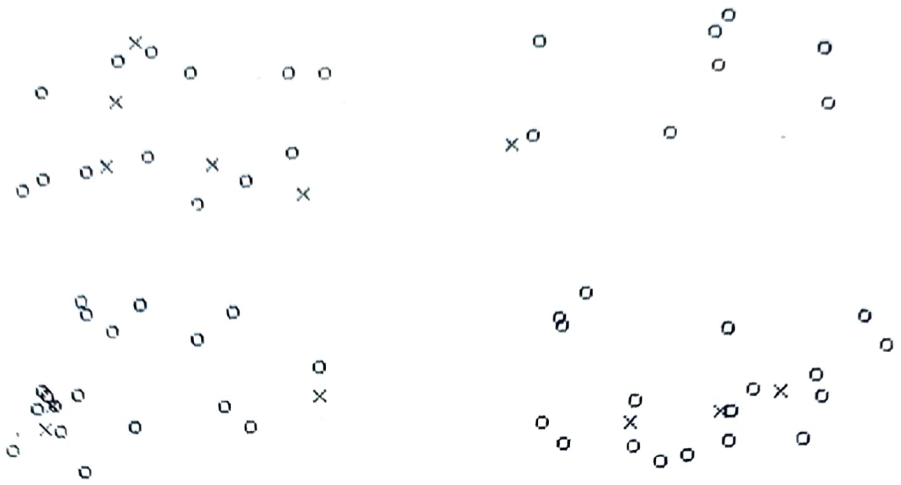

Figure 3. Generation 30.

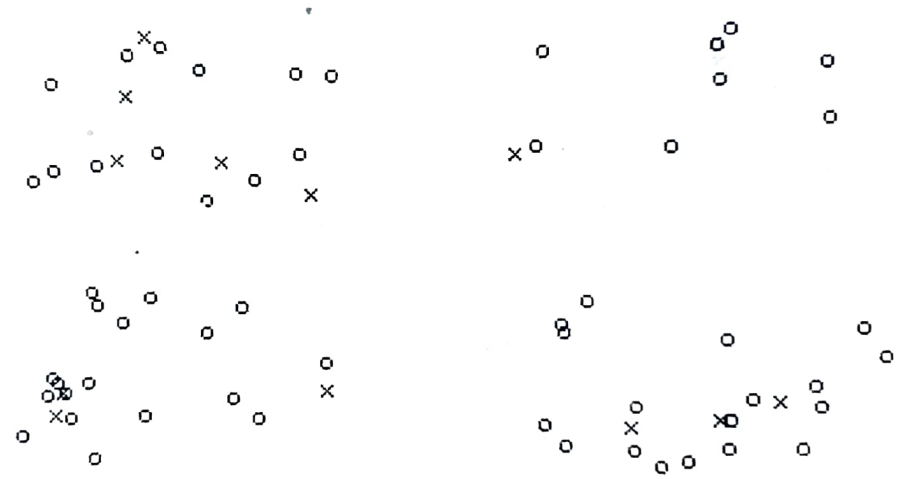

Figure 4. Generation 45.

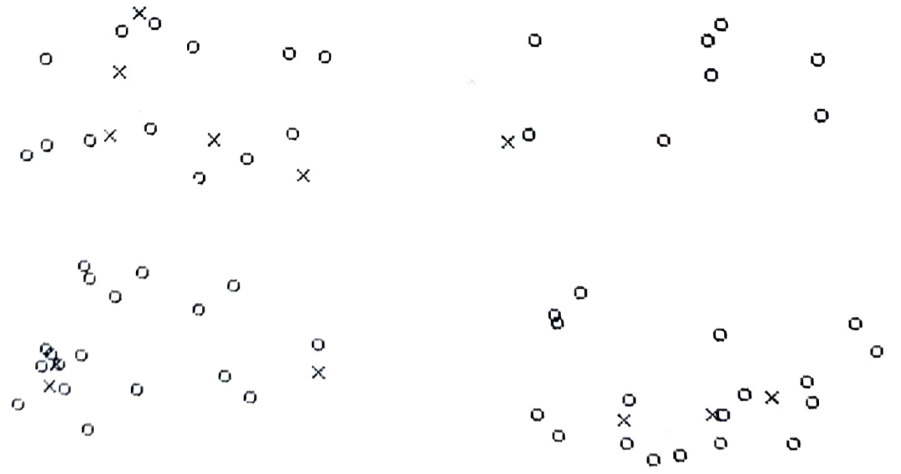

Figure 5. Generation 60.

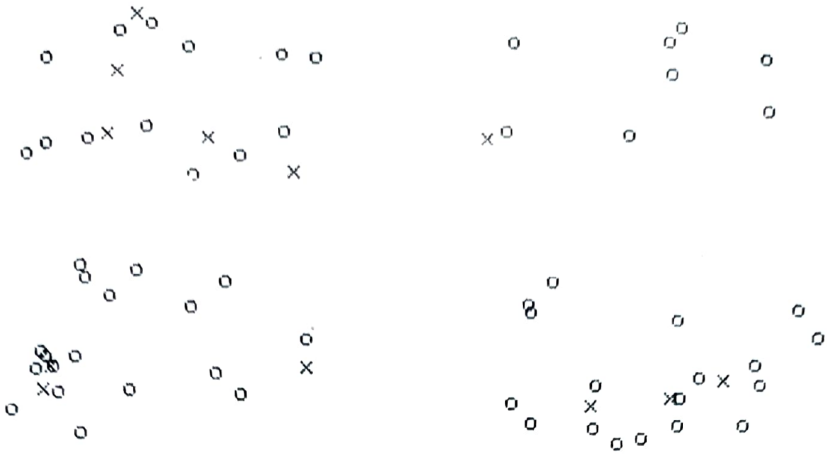

Figure 6. Generation 75.

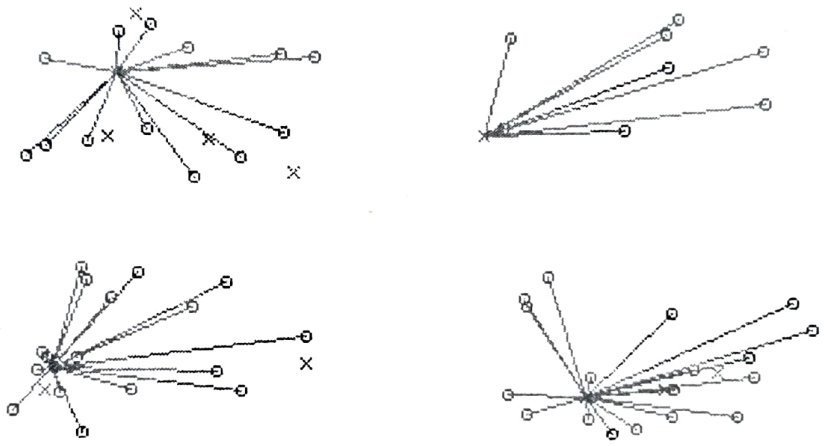

Figure 7. Last generation.

REFERENCES

Bäck, T. , Schwefel, H. P. (1993), An overview of evolutionary algorithms for parameter optimization, *Evolutionary Computation*, 1, 1-24 .

Bäck, T. , Rudolph, G. , Schwefel, H. P. (1993), Evolutionary Programming and evolution strategies: similarities and differences, in Proceedings of the Second Annual Conference on Evolutionary Programming, D.B. Fogel, W. Atmar (editori), *Evolutionary Programming Society, La Jolla,* 11-22 .

Fogel, D. B. (1992), Evolving Artificial Intelligence, Ph. D. Thesis, *University of California, San Diego* .

Fogel, D. B. , Atmar, J. W. (editori) (1992), Proceedings of the First Annual Conference on Evolutionary Programming, *Evolutionary Programming Society, La Jolla.*

Fogel, D. B. (1994), An introduction to simulated evolutionary optimization, *IEEE Transaction Neural Networks*, 5.

Fogel, D. B. (1995), Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, *IEEE Press, Piscataway, NJ* .

Fogel, L. J. (1962), Toward inductive inference automata, Proceedings of the International Federation for Information Processing Congress, *Munich,* 395-399 .

Fogel, L. J. (1964), On the Organization of Intellect, Ph. D. Thesis, *UCLA*.

Fogel, L. J. , Owens, A. J. , Walsh, M. J. (1966), Artificial Intelligence through Simulated Evolution, *Wiley, New York*.

Babeş-Bolyai University, Faculty of Mathematics and Informatics, RO 3400 Cluj-Napoca, str. Kogalniccanu 1, România.

E-mail address: {davram,ddumitr}@cs.ubbcluj.ro

Dipartimento di Ingegneria della Informazione, Universita di Pisa, Italia

E-mail address: beatrice@iet.unipi.it