STATISTICA, anno LXIX, n. 2–3, 2009

EVOLUTIONARY COMPUTATION METHODS AND THEIR APPLICATIONS IN STATISTICS

Francesco Battaglia
Department of Statistics, Sapienza University, Rome, Italy.

1. Evolutionary computation between artificial intelligence and natural evolution

Drawing an history of evolutionary computation is not an easy task, we shall spend only some words to outline where and when the framework, which has later become that of evolutionary computation, originated.

The term "evolutionary" started to be associated with concepts linked to computing and algorithms towards the beginning of the sixties, in the scientific community concerned with developing and studying computing machines, and specifically in a field which has been known as *artificial intelligence*. An excellent discussion may be found in Chapter 1 of Fogel (1998). The distinctive feature of changing continuously the behavior in order to obtain benefits according to a given measure of satisfaction has been advocated as an essential characteristic of intelligence, and provides the link between artificial intelligence and evolutionary computation.

A similar link may be drawn in nature between intelligence and adaptation. In effect, it is doubtless that intelligence arises when an organism reacts to the external forces (say, the *environment*) and many reactions are possible, with different consequences. Intelligence may be thought of as related to choosing reactions that ensure best results and it in turn calls for a definition and a measure of reaching goals, or satisfaction. Thus, the concept of intelligence requires, on one hand, that different courses of action may be alternatively chosen, in other terms, is concerned with decision maker entities, and, on the other hand, requires that each possible action may be evaluated, in order that good and bad behaviors may be distinguished. But the outcome of each action depends generally on the environment, therefore intelligence requires that information from the environment may be recovered and analyzed. Fogel *et al.* (1966) state: "intelligence is the capability of a system to adapt its behavior to meet its goal in a range of environments". In a word, though extremizing: *intelligence is evolution*.

The last word, evolution, introduces the second framework which inspired the development of evolutionary computation methods, and from which they inherited the name itself, the terminology and the fundamental ideas: the theory of natural evolution, named after Charles Darwin.

The main results obtained by Darwin in the middle of the nineteenth century received confirmation later and further development from the progress of genetics, and now a systematic theory of natural evolution has been formulated and its validity recognized almost everywhere, also known as *neo-Darwinism*. The dynamics of life is explained through few interaction mechanisms among individuals, and between individuals and the environment. Each of such mechanisms is subject to random forces, and their outcome is stochastic.

The first process is reproduction, where each individual, or each pair if sexual reproduction is considered, bears new individuals that have some features of their parents, and can mix characters of their two parents in sexual reproduction. Mutation is the second process, which arises also in the transition from parents to children, and allows new characters to appear in the population. The competition mechanism implies that individuals in the same population, owing to finite resources, are antagonists and compete to ensure sufficient (or satisfying) benefits, and implies also that some individuals are successful while others are not and are, at various extent, eliminated from the population. Finally, the selection process dictates that not all individuals have the same reproduction ability, and such differences are related to the capacity of the individual to adapt to the environment and survive to the competitors and predators.

Given a population at a certain time (the initial generation), a new population (the new generation) is formed, after all such processes have acted on each individual of the initial population. Some individuals of the initial population survive to the new population, and some new individuals are born. Modifications are random but the essential feature is that individuals that are more adapted (fitted) to the environment tend to survive and reproduce to a larger extent than individuals that are less fitted, therefore favorable characteristics tend to spread into populations, generation after generation.

Once the fundamental processes of natural evolution were understood, it became possible to formalize them into a mathematical algorithm: the *Genetic Algorithm*, though only relying on simplifying assumptions, because such processes are too complex.

The description of genetical evolution given above is very approximate and imprecise and does not approach in any detail the physical and chemical processes involved: really, we have not mentioned DNA at all. However, it is rich enough to be suitable for most purposes, and indeed the best mathematical metaphor of the evolution process, the genetic algorithm, is based on an even more simplified scheme.

In a genetic algorithm, each individual is assigned only one chromosome which characterizes all its features (possibly including several distinct fragments), and most often the information carried by each gene is simply binary. Moreover, there is no sex distinction, thus any individual may mate with any other to bear an offspring; finally, no growing up or ageing exist, and the fitness of each individual is constant in time. In spite of its simplicity, the genetic algorithm proved a formidable, all purposes optimizing tool; its simplicity explains also why an overwhelming number of variants and generalizations have been proposed in the literature.

Genetic algorithms originate from the work of John Holland in the sixties, and were developed by him and his students at the University of Michigan during several years. The fundamental reference, where genetic algorithms are formally introduced and thor-

oughly explored, is Holland's book *Adaptation in Natural and Artificial Systems* printed in 1975. As is clear from the title, Holland's idea was to formalize and describe a quantitative framework for studying the process of adaptation and evolution of a biological population, and to learn rules and principles which could allow artificial systems, in a similar fashion, to evolve. The latter are often related to optimization problems, thus employing the genetic algorithm environment for solving optimization problems has obvious advantages.

Each individual of the population is associated to a possible solution of the problem, and the function to be optimized assumes the meaning of fitness. Therefore, the fittest individual in each generation represents the best solution reached thus far, and evolution allows to discover a solution that gets better, as the generations evolve. In this way, running a genetic algorithm allows to produce a sequence of solutions (the best fitted individuals at each generation) which approach the optimum.

Genetic algorithms have been successfully employed for solving optimization problems in many fields, from physics to engineering to medicine; we shall address here only some applications in Statistics.

What all these problems have in common is complexity. In effect, there is no point in trying to solve by means of a genetic algorithm a problem whose solution may be found by means of analytical methods (such as equating derivatives to zero), or simple and well-behaved problems where classical numerical techniques such as Newton's method are generally adequate. Therefore genetic algorithms are employed in problems for which, essentially, no method for determining the optimum is known better than enumerating all the possible solutions. Such problems are known in the computational complexity theory as NP-complete problems.

The last 30 years have seen an enormous development of the theory originated by John Holland, bridges have been built between the several similar attempts linked to the idea of evolution, and now a generally accepted framework called *evolutionary computation* has been established; however, genetic algorithms still appear the most general and flexible tool for evolutionary computation, and often evolutionary computing and genetic algorithms are used as synonymous.

2. Evolutionary computation methods

Though evolutionary computation is not only optimization, most applications in Statistics are concerned, at least at present, with optimization problems.

Schematically, an optimization problem may be defined as a pair (f, Ω) where f is a function from Ω to the set of real numbers $\mathbb{R}$, and Ω is the set of possible solutions. If Ω is a subset of $\mathbb{R}^n$ but is not a subset of $\mathbb{R}^{n-1}$, then n is the dimension of the problem. The aim may be to maximize or minimize the value of the objective function f. Solving the problem means finding the element(s) of Ω for which f attains its maximum (minimum) value, if they exist.

In many lucky (or simplified) cases, f is a well-behaved mathematical function of n real arguments, and Ω is a sufficiently regular subset of $\mathbb{R}^n$ so that the solution is found simply by equating to zero the derivatives of f. But in the great majority of

cases this is not possible, the most relevant instance in Statistics is when Ω is a discrete, though very large, set. In that case we speak of combinatorial optimization problems. In practice such problems can be exactly solved only enumerating, and evaluating, all possible solutions (all elements of the set Ω) and this becomes rapidly impossible as n increases. Thus, we have to be satisfied with approximate solution methods, trying to build algorithms which enable to obtain "sufficiently good" solutions in a reasonable time: such methods are called *heuristic* algorithms.

An exact and widely shared definition of heuristic algorithm does not exist, but a generally accepted idea is that a heuristic method searches for best solutions of a problem at a reasonable computational cost, without ensuring to reach optimality, and consists in an iterative sequence of moves inside the solution space towards better points, trying to avoid evident errors.

Most heuristic methods are based on local search algorithms, in the sense that they wander inside the solution space, and each move is determined by evaluating neighboring candidates, and entering those which are considered more promising according to pre-specified given criteria.

Since heuristics may profit of the particular and specific features of each single problem, it is likely that ad-hoc techniques be suitable, and that a heuristic that performs well on a kind of problem does not work so good in another. A series of theorems (called *no free lunch theorems*) originated by the work of D. H. Wolpert and W. G. Macready (see e. g. Wolpert and Macready, 1997) state, essentially, that, if averaged on sufficiently broad classes of problems, any pair of heuristic algorithm give equivalent results. Therefore, heuristic methods are often tailored to the specific problems they have to solve.

There are, however, some all-purposes stochastic search methods which rely on basic ideas useful for any optimization problem, and for this reason are called *meta-heuristic*.

A heuristic optimization algorithm may be essentially described as an iterative method which defines a sequence of elements in the solution space, with the aim of progressively improving the values of the objective function. It is based at least on the following items:

(a) A set of possible solutions Ω

(b) A function $f(.)$ defined on Ω and taking values on the set of real numbers (or some subset), called the objective (or target, or score) function, and a scope: maximization or minimization

(c) A set of neighborhoods defined for each element of Ω, or a rule for describing the neighboring elements of each point $\omega \in \Omega$, say $\{N(\omega), \omega \in \Omega\}$.

(d) A set of moving rules, which determine, for each $\omega \in \Omega$, which element in the neighborhood $N(\omega)$ is chosen as the next solution to be evaluated. Rules are often random, in that case they specify a probability distribution on the elements of the neighborhood $N(\omega)$.

(e) A starting point $\omega_0 \in \Omega$

(f) A stopping rule for deciding when the iterative process should be terminated, and the best-so-far found solution to be declared the result of optimization.

The heuristic is iterative: starting from ω_0, it computes its neighborhood $N(\omega_0)$ according to (c), chooses a point $\omega_1 \in N(\omega_0)$ according to (d), and turns from ω_0 to ω_1; then the elements of $N(\omega_1)$ are examined and one of them, ω_2, is selected; then $N(\omega_2)$ is computed and a new element ω_3 in it is selected and so on. If the stopping rule is met at iteration n, then ω_n is assumed as the solution and $f(\omega_n)$ as the optimized value of the objective function.

If the decisions concerning items (c), (d), (e) and (f) do not depend on the particular behavior of the objective function $f(.)$, but are specified in terms of mathematical procedures defined on the formal objects $f(.)$ and Ω, and may therefore applied to any particularization of them, the algorithm is called a *meta-heuristic*.

Many different heuristic methods have been proposed and are employed in various fields, their differences are essentially in the way the neighborhoods, and, more important, the moving rules, are selected, the most relevant alternatives being between deterministic and stochastic procedures, and between monotonic — $f(\omega_{k+1})$ always better than $f(\omega_k)$ — or non-monotonic rules.

Relevant examples of meta-heuristic methods are the descent methods, threshold accepting, simulated annealing, tabu search.

Evolutionary computation methods share the common features of meta-heuristics, but in addition have specific characteristics:

1. Evolutionary computation methods are population-based algorithms. It means that they consider at each stage an entire subset of possible solutions as the individuals of a population, and at each iteration (in this framework called *generation*) each member of the population changes (or only some of them); furthermore, the number of individuals may also change iteration by iteration. The essential feature, however, is that evolution originates from reciprocal interactions among individuals, in other terms the choice of which individual will become part of the population at generation n depends on the whole set of individuals in generation $n-1$.

2. The moving rules are stochastic. Therefore, at each step, each individual is assigned a neighborhood and a probability distribution on that, and the individual is changed to a new one inside the neighborhood according to a random trial on that probability distribution.

In summary, and reconciling the naturally inspired terminology of the previous Section with the more mathematical style of this Section, an evolutionary computation method may be described as follows:

(a) The algorithm motivation is to search for the maximum value of a given function $f : \Omega \to \mathbb{R}$. Each element of the set Ω is called an individual, and the value of $f(.)$ measures the success of that individual in its environment, therefore f is called the fitness function and the aim is to maximize it.

(b) The algorithm is iterative and at each stage n (called a generation) produces a subset of Ω, called the population at generation n. A rule for choosing the population at the initial stage (generation 0 population) is therefore needed, and also a stopping rule for determining at which generation the iterations should terminate, has to be selected.

(c) The composition of population at generation n is determined by the population of the previous generation according to a stochastic procedure, which is inspired by the natural evolution theory, and tries to translate into formal mathematical operations the biological processes of reproduction, mutation, recombination. Members of generation $n-1$ are seen as parents, and those of generation n as children. The number of individuals in the population may remain constant, increase or even decrease at each iteration (according to the different evolutionary computation algorithms).

The essential innovation of population based methods is that the "destiny" of each individual at generation n depends on the whole set of individuals in the population at that time; this means that for each individual $\omega \in \Omega$ its neighborhood, defining what it will become in the next generation, depends on the other individuals of the contemporary population. Furthermore, different individuals in the same generations may have different types of neighborhood. Thus, the most common way of analyzing and following the evolution across generations does not refer to neighborhoods of each individual, but rather focuses on offsprings arising at each generation from the population as a whole.

Since the transition from a generation to the next one has a complicate mechanism, it may be generally decomposed into different successive phases. The first one consists in selecting, from the individuals of the population at the current generation, a set of candidates for bearing new offsprings (a stage often referred to as selection); then, each of them (or more frequently each pair) undergoes a process consisting in the successive application of some modification rules (the evolutionary "operators", frequently inspired by analogies with nature), ending up in the creation of one (or sometimes two) new individual (the child). Finally, children may replace their parents (totally or partly or not at all according to different replacement schemes), and constitute the population for the next generation.

A classification of evolutionary computation method is not simple, and many different criteria have been proposed. However, at least on an historical ground, there is a general agreement that three methods: evolutionary programming, evolution strategies and genetic algorithms, originated the family of evolutionary computation algorithms. For an account of evolutionary programming, the reader is referred to Fogel *et al.* (1966) and Koza (1992). We describe evolution strategies and genetic algorithms briefly in the next sections. More recently, many new methods have been proposed and explored. Among them, we shall address only a few methods which appear more suitable for applications in Statistics: the estimation of distribution algorithm, the differential evolution and some evolutionary behavior algorithms.

2.1. Evolution strategies

An optimization approach which is also based on the idea of evolution was proposed in the sixties by I. Rechenberg and H. P. Schwefel at Berlin Technical University. Here the motivation comes from a process control problem in an engineering framework, and was soon extended to optimization of general functions of multiple variables. A detailed account may be found in Back (1996), and a more concise and recent introduction in Beyer and Schwefel (2002).

In its original formulation, an evolution strategy is an algorithm for optimizing a real valued function of M real variables: $f : \mathbb{R}^M \to \mathbb{R}$. Evolution strategies also are based on populations of individuals, which evolve through successive generations. An individual is a possible solution to the problem, and therefore is identified by a vector in $\mathbb{R}^M$. The fitness evaluation is obviously made through the function to be optimized f (thus proportional to f if the maximum is needed, and to $-f$, or inversely proportional, if the minimum is required).

The algorithm is iterative and the population at the beginning is composed by individuals chosen at random, uniformly, in the whole space of solutions. At each generation, offsprings are generated by selecting a parent and perturbing it by simply adding a random realization of a $M-$variate gaussian variable with zero mean and a given (diagonal) dispersion matrix. Formally, from the individual $x = (x_1, x_2, \ldots, x_M)'$ the offspring $y = (y_1, y_2, \ldots, y_M)'$ is obtained from

$$y_i = x_i + z_i \sigma_i \quad i = 1, 2, \ldots, M \tag{1}$$

where $(z_1, z_2, \ldots, z_M)$ are independent realizations of a gaussian N(0,1) variable, and $(\sigma_1, \sigma_2, \ldots, \sigma_M)$ are pre-specified values.

Once all offspring are created, the population for the next generation is formed by selecting only the fittest individuals, according to different strategies, called $(\mu + \lambda)$ or (μ, λ), where μ is the (constant) number of individuals in the population, and λ in the number of offsprings at each generation:

$(\mu + \lambda)$**–ES** : μ parents are employed for creating λ children, then all $(\mu+\lambda)$ individuals are ranked and the best μ are chosen to constitute the next generation population

(μ, λ)**–ES** : λ offsprings are generated by μ parents, $\lambda > \mu$, and only the μ best fitted out of the λ offsprings are returned for the next generation population: therefore each individual disappears at the next generation.

A number of important extensions and generalizations were introduced in later years, so that the actual evolution strategies employed in recent applications are much more elaborated. Main evolutions of the original strategy concern *self-adaptation* and *recombination*.

Self-adaptation means that the parameters governing mutation are also evolved together with individuals. Therefore each member of the population is characterized not only by its numerical vector related to the solution, but also by the vector of the standard errors of the perturbations: an individual is coded by $(y_1, \ldots, y_M, \sigma_1, \ldots, \sigma_M)$. The

standard deviation parameters σ_i are also inherited by the offsprings, and are subject to mutation too. The mutation operator for σ is often described by:

$$\sigma_{new} = \sigma_{old} \exp\{\tau z\}$$

where z is a realization of a gaussian standard variable and τ is a fixed constant (called the learning parameter).

Recombination was proposed in order to get offspring that share characteristics of more than one parent solution. The corresponding evolution strategies, indicated with $(\mu/\rho, \lambda)$–ES or $(\mu/\rho+\lambda)$–ES, rely on a fixed number ρ chosen in advance and called the mixing number ($\rho < \mu$) which defines the number of parents involved in an offspring. For each offspring to be born, ρ parents are selected at random from the population, let $x_i(k), i = 1,\ldots,M; k = 1,\ldots,\rho$ be the solution coordinates of the selected parents: then each coordinate y_i of the offspring depends on the set of the corresponding coordinates of its parents $\{x_i(k), k = 1,\ldots,\rho\}$, and two alternative recombination operators may be adopted:

- discrete (or dominant) recombination: y_i is selected at random with equal probabilities from $\{x_i(1), x_i(2),\ldots,x_i(\rho)\}$, for each $i = 1,2,\ldots,M$.
- intermediate recombination: y_i is chosen equal to the arithmetic mean of the parents coordinates:

$$y_i = \frac{1}{\rho}\sum_{k=1}^{\rho} x_i(k)$$

Evolution strategies are more naturally employed when the space of the solutions is $\mathbb{R}^M$ or at least a compact subset, but they have been modified and generalized to cope also with discrete spaces or even with mixed, or constrained spaces, though it may impose several restrictions on the mutation and recombination operators.

2.2. *Genetic algorithms*

Among the evolutionary computation methods, genetic algorithms are those which most rely on biological inspiration and try to develop more closely a metaphor of the natural evolution of biological populations. The terminology itself bears many analogies with genetics and biology.

A genetic algorithm is an iterative procedure that follows the evolution of a population of individuals through successive generations . The features of each individual are formalized in a vector of symbols from a given alphabet (usually a binary or decimal digit) called the *chromosome*. Each entry of this vector is called a *gene*. Not only the value of each gene, but also its position (the *locus*) in the chromosome is relevant in defining the characteristics of the individual. The whole set of characteristics of each individual determines its ability to survive and reproduce in the environment, and this is supposed to be measurable by means of a positive number, called the fitness of the individual. Thus, a positive function is defined on the set of chromosomes, which measures the fitness and is called the fitness function.

We shall denote by g the index of the generation, by i the index of the individual in the population ($i = 1,2,\ldots,N$) and by j the index of the gene in the chromosome ($j = 1,2,\ldots,M$) . Therefore the individual i at generation g is associated to the following chromosome :

$$x_i^{(g)} = (x_{i,1}^{(g)}, x_{i,2}^{(g)}, \ldots, x_{i,M}^{(g)})'$$

and $f[x_i^{(g)}]$ will denote its fitness value.

Transition from a generation to the next consists in a reproduction process, of a stochastic nature, articulated in the three stages of selection, recombination and mutation.

The selection step results in choosing which individuals of the current population are going to reproduce. In agreement with the concept of natural evolution, we want that most fitted individuals reproduce more frequently than less fitted ones, therefore, we set up a random procedure where the probability of being selected for reproduction is an increasing function of the fitness value. Most popular rules (or selection operators) are the *roulette wheel*, the *stochastic universal sampling*, and the *tournament selection*.

The roulette wheel method simply selects an individual with probability proportional to its fitness. Therefore, each choice of a candidate to reproduction is made by a random trial with possible results $\{x_i^{(g)}, i = 1,\ldots,N\}$ and associated probability

$$P[x_i^{(g)}] = f[x_i^{(g)}] / \sum_{k=1}^{N} f[x_k^{(g)}].$$

The roulette wheel rule may be also seen in terms of the fitness empirical distribution function $F_g(y) =$ Freq {individuals with fitness $< y$ at generation g }, and amounts simply to choosing independently N times a number r uniformly distributed between 0 and 1, and selecting the corresponding r-quantile of F_g (i.e., the individual that has the largest value of F_g less than r).

Stochastic universal sampling, on the contrary, is obtained by generating uniformly at random only one number ℓ in $(0, 1/N)$, and choosing the individuals corresponding to quantiles $\ell, \ell + 1/N, \ell + 2/N, \ldots, \ell + (N-1)/N$ of F_g.

Tournament selection is a method which tries to exploit further similarities with natural populations, and is based on comparisons of individuals (a tournament) where the best fitted wins. For each candidate to reproduction to be selected, a group of individuals is chosen at random (but with different modalities according to variants of this method), they are compared and the one with the largest fitness is selected. The replacement actually takes place with probability p_s (*selection pressure*).

Once the selection stage has produced candidates for reproduction, the recombination stage considers randomly chosen pairs of individuals. They mate and produce a pair of offsprings that may share genes of both parents. This process, also called *cross-over*, is applied with a fixed probability (usually larger than 0.5 but smaller than one) to each pair. Several different types of cross-over are common, the simplest is called one point cross-over. It consists in pairing the chromosomes of the two individuals and

choosing at random one locus: the genes which appear before that locus remain unchanged, while the genes appearing after the cross-over point are exchanged together. The process produces, from two parents, two children, each of which inherits part of the gene sequence from one parent, and the remaining part from the other parent. If cross-over does not take place, the two children remain identical to their parents. A more elaborated cross-over type is called uniform cross-over: each gene of the offspring is selected at random, from the corresponding genes of the two parents, with equal probability. Note that in this way the number of offsprings from a pair may be chosen to be one, two or even more.

Once offsprings are generated, they are subject to the mutation operator. Mutation is needed to introduce innovation into the population (since selection and cross-over only mix the existing genes), but is generally considered a rare event (like it is in nature). Therefore, a small probability p_m is selected, and each gene of each individual's chromosome is subject to mutation with that probability, independently of all other genes. If the gene coding is binary, a mutation simply changes a 0 to a 1 or vice versa, while if the alphabet for a gene is richer, a mutation rule has to be defined. Often a uniform random choice among all possible symbols (different from that to be mutated) is preferred if the alphabet is finite, or a perturbation based on a gaussian variable if the genes are coded as real numbers.

A final important topic is the way new offsprings replace their parents, also called reproduction (sometimes replacement) strategy. We assume that the number of individuals in the population for each generation remains equal and fixed to N; the simplest rule is generating N offsprings and replacing entirely the population at each generation. However, this way may eliminate the best fitted individual with non zero probability, therefore a common modification is as follows: if the best fitted offspring is worse than the best fitted parent, this last is retained in the next generation population, usually replacing the least fitted offspring. It is called the *elitist* strategy . More generally, we may decide to replace at each generation only a fraction (called the *generation gap*) of the population, or even to replace just one individual (usually the worst) at each generation (a procedure called *steady state* or *incremental rule*).

Many modifications of the genetic algorithm have been proposed in literature, and several types of different procedures concerning mutation, recombination and replacement have been introduced. Furthermore, some new operators have been considered, for example *inversion* (which consists in reversing the order of the genes in a chromosome) that are employed less frequently.

Genetic algorithms start from an initial population (generation 0) whose individuals are usually selected at random uniformly in the solution space, or, sometimes, are chosen in a deterministic fashion in order to represent different subsets of that space. As generations proceed, new individuals appear: owing to the selection mechanism, their fitness is on the average larger than their parents; moreover, the cross-over operator allows appearance of new individuals that may enjoy favorable characteristics of both parents. Thus, it is likely that after many generations the best chromosome that may be obtained by combining the genes of the initial population is discovered and starts to replicate itself; however, this process would be unable to experiment new genetic ma-

terial. In other words, in case that all individuals at generation 0 have say a gene code equal to 1 in the first locus of their chromosome, this would be true for any offspring, indefinitely, and if the maximum fitness corresponds to a chromosome whose first gene equals 0, this would never be discovered. The solution to this drawback is obviously mutation: such operator ensures that all possible gene sequences may be obtained in the chromosome of the offspring, and therefore, in principle, we may be confident that running a genetic algorithm for a sufficiently large number of generations the space of solutions is thoroughly explored, and the best individual found. Formal convergence properties of the genetic algorithm have been analyzed in depth, see Reeves and Rowe (2003) for reference.

2.3. Estimation of distribution algorithms

These algorithms are best explained, and were originally derived, in the case that the chromosomes are real vectors $x = (x_1, x_2, \ldots, x_M)'$, though they have been extended to more general settings. In the real vector case, the problem may be formulated as that of maximizing a fitness function $f(x)$ where x is a real vector $x \in \mathbb{R}^M$.

The proposal originates from the attempt of explicitly taking into account the correlation between genes of different loci (components of the vector x), that may be seen in good solutions, assuming that such correlation structure could be different from that of the less fitted individuals. The key idea is to deliver an explicit probability model and associate to each population (or a subset of it) a multivariate probability distribution.

An initial version of the estimation of distribution algorithm was originally proposed by H. Muhlenbein and G. Paas (1996), and then many further contributions developed, generalized and improved the implementation. A thorough account may be found in Larrañaga and Lozano (2002) and in a second more recent book (Lozano *et al.*, 2006).

The estimation of distribution algorithm is a regular stochastic population based evolutionary method, and therefore evolves populations through generations. The typical evolution process from one generation to the next may be described as follows:

1. Generate an initial population $P^{(0)} = \{x_i^{(0)}, i = 1, \ldots, N\}$; $c = 0$.

2. If $P^{(c)}$ denotes the current population, select a subset of $P^{(c)}$: $\{x_j^{(c)}, j \in S^{(c)}\}$ with $|S^{(c)}| = n < N$ individuals, according to a selection operator.

3. Consider the subset $\{x_j^{(c)}, j \in S^{(c)}\}$ as a random sample from a multivariate random variable with absolutely continuous distribution and probability density $p^{(c)}(x)$, and estimate $p^{(c)}(x)$ from the sample.

4. Generate a random sample of N individuals form $p^{(c)}(x)$: this is the population at generation $c + 1$, $P^{(c+1)}$.

5. If a stopping rule is met, stop; otherwise $c + 1 \to c$ and return to 2.

The originally proposed selection operator was the truncation selection, in other words only the n individuals with the largest fitness out of the N members of the population are selected. Later, it was proposed that other common selection mechanism such as the roulette wheel (proportional selection) or the tournament selection (choice of the best fitted inside a group of k individuals chosen at random) can be adopted.

Note that the most critical and difficult step is the third one: the aim is to summarize the properties of the most fitted part of the population through a probability distribution, and then to employ that distribution to generate new offsprings. Many papers concentrated on the problem of "estimating" the distribution, meaning to derive, from the finite set of individuals $\{x_j^{(c)}, j \in S^{(c)}\}$, the probability density function $p^{(c)}(x)$. It can be observed that this is not an uncommon problem in Statistics, and many proposals (frequently based on non parametric density estimation) appear in the statistics literature. However, the estimation of distribution algorithms were developed in a different field, and methods for deriving $p^{(c)}(x)$ were proposed which are somewhat unusual in Statistics.

The simplest way is called Univariate Marginal Distribution Algorithm (UMDA) and assumes that $p^{(c)}(x)$ is a multivariate normal with independent components, therefore its parameters (means and variances) are obtained by the usual estimators on the marginal distributions of each gene in the subset $S^{(c)}$. Obviously, this is in principle a very inefficient choice, which also contradicts the basic assumption of exploiting correlations among genes, though it has been reported that in some cases it leads to successful algorithms.

A more elaborated solution takes into account the dependence between genes, but to this aim a simplifying assumption is formulated, and is known as Factorial Distribution Algorithm (FDA). The key assumption is that the fitness function may be additively decomposed in terms, each depending only on a subset of genes. In other words, if $s_1, s_2, \ldots, s_k$ all are subsets of $\{1, 2, \ldots, M\}$, it is assumed that there exist k functions $f_1, f_2, \ldots, f_k$, each of which depends on x only through the coordinates corresponding to the subset s_j, and the fitness $f(x)$ may be written as follows:

$$f(x) = \sum_{j=1}^{k} f_j[x(s_j)]$$

where $x(s_j) = \{x_i, i \in s_j\}$. If it is true, the multivariate probability distribution $p^{(c)}(x)$ is factorized as a product of conditional distributions on smaller subsets, and this simpler factorization helps to estimate the whole distribution. The choice of the subsets may be done also using graphical models.

More complicated ways of estimating the multivariate distribution have also been proposed, based on Bayesian Networks.

As should be clear, the accent in estimation of distribution algorithms is essentially on extracting the typical features of best fitted individuals, and reproducing them in the next generation. Since the use of the complete multivariate probability distributions accounts for relationships between genes, no recombination tool (cross-over operator) is contemplated. Neither mutation operators are used: new individuals (carrying never

experienced gene values) may be generated by random sampling the estimated multivariate probability distribution. In fact, since $p^{(c)}(x)$ is a density, sampling may result in elements not belonging to $S^{(c)}$.

2.4. Differential evolution

Differential evolution is an algorithm arising in a pure optimization framework, and is best explained by referring to a real function f of M real variables: $f : \mathbb{R}^M \to \mathbb{R}$ to be optimized. Though differential evolution papers refer generally to minimization, we shall continue our biological populations similitude and consider f as a fitness function to be maximized.

Differential evolution is a recent method, which was first proposed by R. Storn and K. V. Price in 1995. A complete account may be found in Price *et al.* (2005).

Differential evolution is also a population based method, and evolves populations of solutions through successive generations. Each individual represents a possible solution, and is codified by a real vector $x = (x_1, x_2, \ldots, x_M)' \in \mathbb{R}^M$. Though not frequently used in the present framework, we continue to denote x by the word chromosome, and their components by the word gene.

The transition from one generation to the next is obtained by treating each individual separately, and producing an offspring which replaces it in the next generation. This makes differential evolution particularly suitable for a parallel implementation.

The evolution mechanism is based on *difference vectors* (the difference between two randomly selected chromosomes of the current population) which is the basic perturbation type allowing new features to appear in the population. The word vector is usually preferred to chromosome, because differential evolution has a convenient geometrical interpretation, and individuals may be advantageously seen as points (or vectors) in $\mathbb{R}^M$.

The number of individuals in the population is held fixed throughout generations; the initial population has a special importance, and the authors stress that good performances of the differential evolution algorithms are critically influenced by the initial population : "in order for differential evolution to work, the initial population must be distributed throughout the problem space" (Price *et al.*, 2005,p. 53). This happens, as it will be soon clear, because the offsprings always lie in the convex closure of the set of the points representing parents population. Therefore, the population at generation 0 should contain points which are as much as possible different with each other, and anyway it should be reasonably ensured that the best solution vector is a linear combination of the vectors forming the initial population. The initial individuals are selected at random by defining a probability distribution on the solutions space, the chosen distribution is usually uniform but different families of distributions are obviously possible.

At each generation, each individual is subject to the process of *differential evolution*, which may be explained as follows. Let x denote the individual to be evolved. A completely different individual is formed as follows: select at random a vector in the population, different from x, and called the *base vector* v_0; also, select at random two more individuals in the population, different both from x and v_0, and different each

other: v_1 and v_2. Scale the difference between these two last vectors by a factor F (the *scale factor*) obtaining $F(v_1 - v_2)$, and add this difference to the base vector, to obtain a new individual u called the *mutant*:

$$u = v_0 + F(v_1 - v_2).$$

The scale factor F has a fixed value chosen by the implementer, usually between 0 and 1. The mutant vector u is then recombined with the initial individual vector x to produce an offspring by means of uniform cross-over. It means that each gene of the offspring will be selected at random to be equal to the corresponding gene of the mutant u with a fixed probability p_c, or equal to the original gene of the individual x with probability $(1 - p_c)$. This random selection is performed independently on all genes. Formally, if $y = (y_1, y_2, \ldots, y_M)'$ denotes the offspring vector, then

$$y_i = \delta_i u_i + (1 - \delta_i) x_i, \quad i = 1, \ldots, M$$

where $\{\delta_i, i = 1, \ldots, M\}$ are independent Bernoulli random variables with equal parameter p_c. The number of genes inherited from the mutant has therefore a binomial distribution.

Finally, a replacement step is performed: the original individual x and the offspring y are compared, and only that with the better fitness is retained and entered in the next generation population. The process of differential evolution is repeated for each individual in the population.

It is apparent that the process induced by differential evolution is logically simple; though the evolution of each member takes place separately, it depends on the whole population because the mutant which may contribute to generate the offspring is obtained by a linear combination of three other individuals selected at random.

A number of different variants and modifications are possible, in various directions concerning the choice of both the initial population and the base vector, and finally the scale factor.

Like all other evolutionary computation methods, differential evolution is iterative and the best individual in the population approaches the optimum as the generations flow; therefore, a stopping rule has to be selected. Owing to the geometrical interpretation, it would appear natural to stop generations when the points representing all members of the population appear nearly undistinguishable, however this problem is not specific to differential evolution, but has similar characteristics for all evolutionary computation algorithms, and any stopping rule is plausible.

2.5. Evolutionary behavior algorithms

Many more optimization algorithms inspired by nature have been proposed in recent years. Among them, some methods are particularly interesting, also population based, but suggested by analogies with the social behavior of the individuals rather than biological reproduction features. Rather than concentrating on the evolution of a population through generations, these methods consider each individual as an agent searching iteratively for the solution of a given problem (equal for all of them). At each stage, each

individual proposes a solution, and the stages are iterated until a stopping criterion is met. The crucial point is that the solution proposed at each stage by each individual depends on the goodness score (or fitness) of the solutions proposed in the previous stage by all (or some) other individuals in the population. Therefore the solution proposed by each individual depends on the best results reached by the community. These methods try to idealize and reproduce the social behavior features of living communities, thus we have called them *evolutionary behavior* algorithms.

The main contributions in this field are the *ant colony optimization* and the *particle swarm optimization*.

Ant colony optimization was introduced by Dorigo in his ph. d. thesis in 1992 (Dorigo, 1992) and was developed by himself and his co-workers in the last fifteen years (e. g., Dorigo and Gambardella, 1997; Dorigo and Stützle, 2004). The method is inspired by the way ants search for food. Each ant initially wanders searching for food, but upon finding it, the ant returns to its colony laying down, in the way back, a substance called *pheromone*. Later on, ants tend to follow pheromone trails rather than wander at random, therefore advantageous paths are characterized by a large amount of pheromone, and are more attractive. However, pheromone evaporates in time, so that paths to places where food has been exhausted lose their attractiveness and are soon neglected.

Ant colony optimization's most natural application is to problems where any possible solution may be described by a path on a graph, like in the traveling salesman problem, and it will described here in that framework, though these methods have been applied also to different environments.

Let us consider a graph with n vertices denoted by $1,2,\ldots,n$ and edges $(i,j) \in E$, and suppose that any possible solution is associated to a path, possibly with some constraints. Each edge has a known cost c_{ij} (often proportional to its length if applicable) and a pheromone loading τ_{ij} (initially set to zero). We consider a population of m ants, and the algorithm is iterative: at each stage each ant proposes a solution, by building an admissible path on the graph. Each ant starts from a randomly selected vertex and chooses which vertex to reach by means of a probabilistic rule: if the ant is in vertex i, it will select vertex j with probability proportional to $\tau_{ij}^{\alpha} c_{ij}^{-\beta}$, where α and β are positive constants. Depending on the nature of the problem, constraints on the edge set E and on the single solution components, and a rule for deciding when a solution is complete, have to be fulfilled (e. g., in the traveling salesman problem each vertex has to be visited once and no more than once).

When each ant k has completed its proposed solution s_k ($k = 1,2,\ldots,m$), these are evaluated by means of a fitness function $F(s)$, and the pheromone values of each edge are updated:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho \sum_{s \in S*} F(s)$$

where ρ, between 0 and 1, controls the evaporation rate, and the sum over s is extended to all solutions including the edge (i,j) contained in a subset of solutions S^* ; in the simplest implementation S^* is the set of the solutions proposed by each ant in the current stage, as in ant systems (Dorigo, 1992). Other ant colony optimization algorithm

differ essentially for the way S^* is built, therefore the differences are in the pheromone updating rule: one may add to S^* the best so far found solution, or on the contrary S^* may contain only the best solution found so far, or just in the current stage, or S^* may contain only the solutions found by the most successful ant at each stage. Finally, the pheromone values may be limited by a priori chosen minimum and maximum values (max-min ant systems, Stützle and Hoos, 2000). The fitness $F(s)$ may also be chosen in various ways, the most common seems an inverse proportionality to the sum of the costs of all edges in the solution.

Particle swarm optimization is a more recent set of algorithms which tend to reproduce a sort of social optimization induced by interactions among individuals with a common problem to solve, also known as *swarm intelligence*. Particle swarm optimization was introduced by Kennedy and Eberhart (1995, 2001). In its simplest form, a swarm is composed by many particles moving in a multidimensional continuous space, and the particle behavior is recorded at discrete times or stages. At each stage, each particle has a position and a velocity. Suppose that there are m particles moving in $\mathbb{R}^n$, denote by $x_i \in \mathbb{R}^n$ the position of the particle i, and by $v_i \in \mathbb{R}^n$ its velocity: then at the next stage the particle will move to $x_i + v_i$. Each point of the space may be evaluated, according to the problem, by means of a fitness function $F(x)$ defined on $\mathbb{R}^n$. After each stage the particle velocities are updated, allowing the particles change directions, and this is done by exploiting knowledge of the best solution found so far by the particle (local best lb_i), the best solution found so far by all particles (global best gb), and (possibly) the best solution found by a set of neighboring particles (neighborhood best nb_i). In any case, particles tend to be attracted towards the promising points found by the neighbors or the community. The velocity updating equations are linear and of the following type:

$$v_i \leftarrow \omega v_i + c_1 r_1 (lb_i - x_i) + c_2 r_2 (gb - x_i) + c_3 r_3 (nb_i - x_i)$$

where ω is an inertial constant usually slightly less than 1, c_1, c_2, c_3 are constants representing the relative strength of personal, community and neighborhood influence, and r_1, r_2, r_3 are random vectors, introducing a probabilistic perturbation.

Though especially suitable for optimization in $\mathbb{R}^n$, discretized versions for searching over discrete spaces have also been proposed.

Ant colony and particle swarm optimization methods have been employed seldom in statistical analysis applications, but given their increasing development it is likely that they will be soon found useful and appropriate for statistical problems too.

2.6. *Genetic Algorithms and random sampling from a probability distribution*

There has been considerable discussion on the role of genetic algorithms as function optimizers (see De Jong, 1993): it is clear that trying to optimize a function by means of a genetic algorithm means following only the best fitted individual at each generation, therefore focusing only on a particular aspect of the algorithm behavior. But the genetic algorithm may be also employed for evolving a population maintaining diversity, so resulting in a non-degenerate multivariate statistical distribution of the chromosomes.

This observation led to explore connections between evolutionary computation and methods for generating random samples according to a given multivariate probability distribution, that we address shortly here. Though we are dealing in effect with pseudo-random sampling, since we are not drawing real samples, but using numbers generated by algorithms running on a computer, we shall discard the prefix pseudo as usual in the statistical literature.

The problem of generating samples from an assigned distribution is an old one in Statistics, but received a great deal of attention, essentially for the multivariate case, in more recent years, when Bayesian researchers started addressing increasingly complicated problems, where the posterior probabilities do not belong to standard families, and the posterior mean — or other indices — is given by integrals whose primitive is not known. These studies led to the MCMC (Markov Chain Monte Carlo) methods, which have been a major research subject in Statistics in the last twenty years.

A detailed illustration of MCMC methods is beyond the scope of this book (and may be found e.g. in Gilks *et al.*, 1996). Let $\pi(x)$, $x \in \mathbb{R}^p$ denote a multivariate probability distribution whose direct simulation by standard random number generators is difficult. The MCMC techniques generate a vector sequence $\{x_t\}$ in $\mathbb{R}^p$ that may be considered a random realization of a Markov chain with equilibrium distribution equal to $\pi(.)$. To this aim, a proposal distribution $q : \mathbb{R}^p \times \mathbb{R}^p \to [0, 1]$ is defined from which random numbers may easily be generated. At time t, given the state x_t, a random realization y from the distribution $q(.|x_t)$ is generated. The move is accepted, i.e., x_{t+1} is set equal to y, with probability given by $\alpha(x_t, y) = \min\{1, \pi(y)q(x_t|y)/[\pi(x_t)q(y|x_t)]\}$. If the move is not accepted, then $x_{t+1} = x_t$. It may be shown that the chain generated this way is ergodic, and its equilibrium distribution is $\pi(.)$. As a consequence, after an initial "burning-in" period, the generated data are recorded and assumed as a random sample from $\pi(.)$, even though they, strictly speaking, are not obviously a sequence of independent realizations.

This method (known as Metropolis-Hastings) has the main advantage that random numbers generation is needed only from the proposal distribution $q(y|x)$, which may be chosen in such a way that generation is conveniently easy.

It was noted that if $\pi(.)$ is multimodal and with strongly correlated components, the sequence generated by the chain may easily get trapped in a local maximum, and many modifications were proposed to avoid such a drawback. A popular idea is to use many chains in parallel. Several proposals in the literature are aimed at improving the "mixing" ability of the algorithm, i.e., its capability of generating chains which are able to visit exhaustively the whole support (for example, the Metropolis Coupled MCMC of Geyer, 1991, or the Simulated Tempering of Marinari and Parisi, 1992). As soon as the practice of using N parallel chains became popular (these algorithms are sometimes called Population Based MCMC), the idea of exploiting interactions between the different chains arose, and some authors proposed to use techniques similar to the genetic operators mutation and cross-over to evolve the N contemporaneous states of the chains as they would be a population. An important difference here is that one has to use operators that do not destroy the convergence of the chains to the equilibrium distribution π. To this aim the operator has to satisfy the prop-

erty of *reversibility*. Reversibility may be described as follows: for an operator ω which changes x to y with probability $p_\omega(y|x)$, ω is reversible with respect to $\pi(.)$ if $\pi(x)p_\omega(y|x) = \pi(y)p_\omega(x|y)$ for any x, y in $\mathbb{R}^p$. For operators that change pairs of states into pairs of states (as in cross-over), the reversibility property may easily be modified: if ω changes (x_i, x_j) into (y_i, y_j) with probability $p_\omega(y_i, y_j|x_i, x_j)$ then ω is reversible w. r. to π if $\pi(x_i)\pi(x_j)p_\omega(y_i, y_j|x_i, x_j) = \pi(y_i)\pi(y_j)p_\omega(x_i, x_j|y_i, y_j)$ for any x_i, x_j, y_i, y_j in $\mathbb{R}^p$.

A MCMC procedure exploiting genetic operators, called Parallel Adaptive Metropolis Sampler, was proposed by Holmes and Mallick (1998). Liang and Wong (2000, 2001) introduced the Evolutionary Monte Carlo methods, combining also concepts from simulated annealing, and several other papers appeared in the literature, a review may be found in Drugan and Thierens (2004).

As far as the mutation operator is concerned, since coding is usually real, generally a small modification by adding a random normally distributed noise with zero mean and moderate variance is adopted (like in evolution strategies). The probability of mutation, however, is not constant at p_m as before, but is determined by a Metropolis acceptance rule: if x_t is the parent and $x_t^* = x_t + \varepsilon$ is the candidate offspring (the mutant), then the mutation is accepted with probability $\min\left\{1, \pi(x_t^*)/\pi(x_t)\right\}$. This ensures reversibility so that mutation does not destroy the chain ergodicity. Note however that with this mechanism the mutation probability is usually rather large compared with the small values of p_m generally adopted in evolutionary computation.

Cross-over operations may be realized through different operators, but always considering that the ergodic character of the chain should be preserved, therefore the cross-over results should leave the equilibrium distribution unchanged. In fact, if two states x_i and x_j are generated according to $\pi(.)$, the results of a standard cross-over operation between x_i and x_j are not in general distributed exactly according to $\pi(.)$. Two solutions have been proposed: one is subordinating cross-over results to an acceptance rule of Metropolis-Hastings type, the other consists in modifying the cross-over mechanism itself.

Let us consider the first solution. Suppose that two independently randomly chosen states x_i and x_j are subject to a cross-over operator ω which generates offsprings y_i and y_j with probability $\phi(y_i, y_j|x_i, x_j)$. We accept the new pair of states y_i, y_j, substituting them to x_i and x_j, with probability

$$\alpha(x_i, x_j, y_i, y_j) = \min\left\{1, \frac{\pi(y_i)\pi(y_j)\phi(x_i, x_j|y_i, y_j)}{\pi(x_i)\pi(x_j)\phi(y_i, y_j|x_i, x_j)}\right\}.$$

It may be shown that this operator satisfies the reversibility property. It follows that if x_i and x_j are independently distributed according to $\pi(.)$, then the results of cross-over y_i and y_j are also independent and distributed according to $\pi(.)$. The only difficult task could be determining the exchange distribution $\phi(y_i, y_j|x_i, x_j)$. However, for most common cross-over forms, such as fixed one-point and uniform, the distribution satisfies $\phi(y_i, y_j|x_i, x_j) = \phi(x_i, x_j|y_i, y_j)$ thus it disappears in the acceptance probability,

which is simply computed as $\min\left\{1, \pi(y_i)\pi(y_j)/[\pi(x_i)\pi(x_j)]\right\}$.

The alternative way of proceeding is taking as offsprings not directly y_i and y_j, but a suitable transformation which enables maintaining ergodicity. The most common solution of this type is called *snooker cross-over* (being inspired by the snooker algorithm proposed by Roberts and Gilks, 1994, in the MCMC framework). Let, as before, x_i and x_j denote the parents, and y_i and y_j the results of a one-point cross-over operator. The offsprings from x_i, x_j are selected as two new points belonging to the lines joining x_i to y_i and x_j to y_j respectively. They are determined by random trials from the conditional distributions induced by $\pi(.)$ along the lines. Thus, if x_i^c and x_j^c denote the results of the snooker cross-over, then

$$x_i^c = x_i + r_1(y_i - x_i) \;\; ; \;\; x_j^c = x_j + r_2(y_j - x_j)$$

where r_1 and r_2 are generated at random from the probability distributions $g_1(.)$ and $g_2(.)$ defined as follows:

$$g_1(r) \propto \pi[x_i + r(y_i - x_i)]|1-r|^{p-1} \;\; ; \;\; g_2(r) \propto \pi[x_j + r(y_j - x_j)]|1-r|^{p-1} \;\; , \; r \in \mathbb{R}.$$

The proof that the chain remains ergodic in this case is more difficult and will not be given here (see, e. g., Goswami and Liu, 2007).

What precedes suggests a close relationship between MCMC methods and genetic algorithms, and one may expect that the issue of random sample generation according to a given distribution might be developed in a complete genetic algorithms framework. A proposal in this direction is in Battaglia (2001), where a genetic algorithm is introduced for drawing random samples from a given multivariate probability distribution.

3. Statistical applications

Evolutionary computation methods have been employed for solving statistical problems increasingly often in the last twenty years.

Most addressed problems are multivariate, and concern the choice of an optimal solution inside a large discrete space, but evolutionary computation was also applied for point estimation, searching for the maximum likelihood or the optimal value of different score functions.

A brief, non exhaustive account of the most relevant statistical applications is given here, and the reader is referred elsewhere for a wider review (see Baragona and Battaglia, 2009).

A first group of applications is concerned with the selection of variables in regression. Chatterjee *et al.* (1996) used a genetic algorithm, and many contributions followed (e.g., Minerva and Paterlini, 2002; Balcombe, 2005; Kapetanios, 2007). The chromosome is a binary string with length equal to the number of independent variables, and each bit indicates presence or absence of the corresponding regressor. The fitness function may be linked to the coefficient of determination R^2, or the F statistic or its p-value, or a form of penalized gaussian likelihood. A genetic algorithm was employed

also in principal components (Sabatier and Reynés, 2008), in Independent Component Analysis (Wu and Chang, 2002), and in the selection of a graphical model (Roverato and Poli, 1998).

An obvious extension is to model selection in time series. Genetic algorithms were proposed for building autoregressive moving average models (e. g., Gaetan, 2000; Ong *et al.*, 2005; Bozdogan and Bearse, 2003) and the identification of transfer functions (Chiogna *et al.*, 2008). The chromosome specifies, generally with an integer encoding, the orders (and also the active lags in case of subset models), and the fitness is a monotone decreasing function of an identification criterion such as Akaike's Information Criterion, or similar. Recently, evolutionary computation methods have been proposed for identifying non-linear time series models, which are particularly difficult to select. In particular, multi-regime threshold and similar models have been addressed using genetic algorithms (see e. g. Wu and Chang, 2002; Davis *et al.*, 2006; Baragona *et al.*, 2004; Baragona and Cucina, 2008).

Genetic algorithms were found particularly effective for the design of neural networks (e.g. Kim and Shin, 2007). The chromosome encodes the network architecture (number of layers, neurons and their connections), and the fitness is proportional to a measure of the outcome.

A typical problem where genetic algorithms are easily applied and useful is outlier detection. Crawford and Wainwright (1995) address the independently distributed observations case. The identification of outliers in time series by means of a genetic algorithm has been proposed by Baragona *et al.* (2001). In outlier applications the chromosomes are generally binary strings with length equal to the number of observations, and the bits set to 1 identify the anomalous observations. The fitness is chosen dependent on a penalized gaussian likelihood, where the penalizing term is proportional to the number of outliers.

An important field of application of evolutionary computation to statistical problems is cluster analysis, since its combinatorial nature is particularly suitable for genetic algorithms. Several genetic clustering methods have been proposed (see, e. g., Murthy and Chowdhury, 1996; Bandyopadhyay and Maulik, 2002; Tseng and Yang, 2001). The coding depends on the clustering method. For hierarchical methods, often a binary chromosome where each bit is assigned to an observed unit is employed. At any stage, each cluster is split into two new clusters, separating the units with bit equal to one from those to which a bit equal to zero corresponds in the chromosome. In non-hierarchical methods with a fixed number of clusters g, often coding is integer and the chromosome is an integer vector with length equal to the number of observations; each gene — a number between 1 and g — specifies what cluster the corresponding observation belongs to. For a $k-$means cluster process, it has also been proposed to encode directly the centroids of each cluster, so that the chromosomes have k genes and coding is real (and multivariate). The fitness function in genetic clustering is determined by one of the several existing index of cluster validity, based on measures of the within-group and between-group dissimilarity. Recent extensions deal with fuzzy clustering (Maulik and Bandyopadhyay, 2003), multi objective cluster analysis (Handl *et al.*, 2003) and clusters of time series (Baragona, 2001).

Finally, genetic algorithms were recently proposed also for the design of experiments. Each chromosome corresponds to a particular combination of factors, and the fitness function is determined by the outcome of the experiment. Satisfactory results have been reported in complex chemical experiments (see Forlin *et al.*, 2008).

REFERENCES

T. BACK (1996). *Evolutionary algorithms in theory and practice.* Oxford University Press, Oxford.

K. BALCOMBE (2005). *Model selection using information criteria and genetic algorithms.* Computational Economics, 25, pp. 207–228.

S. BANDYOPADHYAY, U. MAULIK (2002). *Genetic clustering for automatic evolution of clusters and application to image classification.* Pattern Recognition, 35, pp. 1197–1208.

R. BARAGONA (2001). *A simulation study on clustering time series with metaheuristic methods.* Quaderni di Statistica, 3, pp. 1–26.

R. BARAGONA, F. BATTAGLIA (2009). *Evolutionary computing in statistical data analysis.* In A. ABRAHAM, A.-E. HASSANIEN, P. SIARRY, A. ENGELBRECHT (eds.), *Foundations of Computational Intelligence vol 3 - Global Optimization (Studies in Computational Intelligence vol. 203)*, Springer, Berlin/Heidelberg, pp. 347–386.

R. BARAGONA, F. BATTAGLIA, C. CALZINI (2001). *Genetic algorithms for the identification of additive and innovation outliers in time series.* Computational Statistics & Data Analysis, 37, pp. 1–12.

R. BARAGONA, F. BATTAGLIA, D. CUCINA (2004). *Fitting piecewise linear threshold autoregressive models by means of genetic algorithms.* Computational Statistics & Data Analysis, 47, pp. 277–295.

R. BARAGONA, D. CUCINA (2008). *Double threshold autoregressive conditionally heteroscedastic model building by genetic algorithms.* Journal of Statistical Computation and Simulation, 78, pp. 541–558.

F. BATTAGLIA (2001). *Genetic algorithms, pseudo-random numbers generators, and Markov chain Monte Carlo methods.* Metron, 59, pp. 131–155.

H. G. BEYER, H. SCHWEFEL (2002). *Evolution strategies, a comprehensive introduction.* Natural Computing, 1, pp. 3–52.

H. BOZDOGAN, P. BEARSE (2003). *Icomp: A new model-selection criterion.* In H. H. BOCK (ed.), *Classification and Related Methods of Data Analysis*, Elsevier Science Publishers (North Holland), Amsterdam, pp. 599–608.

S. CHATTERJEE, M. LAUDATO, L. A. LYNCH (1996). *Genetic algorithms and their statistical applications: an introduction.* Computational Statistics & Data Analysis, 22, pp. 633–651.

M. CHIOGNA, C. GAETAN, G. MASAROTTO (2008). *Automatic identification of seasonal transfer function models by means of iterative stepwise and genetic algorithms.* Journal of Time Seies Analysis, 29, pp. 37–50.

K. D. CRAWFORD, R. L. WAINWRIGHT (1995). *Applying genetic algorithms to outlier detection*. In L. J. ESHELMAN (ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, pp. 546–550.

R. DAVIS, T. LEE, G. RODRIGUEZ-YAM (2006). *Structural break estimation for nonstationary time series models*. Journal of the American Statistical Association, 101, pp. 223–239.

K. A. DE JONG (1993). *Genetic algorithms are not function optimizers*. In D. WHITLEY (ed.), *Foundations of Genetic Algorithms 2*, Morgan Kaufman, San Mateo, pp. 1–18.

M. DORIGO (1992). *Optimization, Learning and Natural Algorithms*. Ph.D. thesis, Politecnico di Milano, Italy.

M. DORIGO, M. GAMBARDELLA (1997). *Ant colony system: a cooperative learning approach to the traveling salesman problem*. IEEE Transactions on Evolutionary Computation, 1, pp. 53–66.

M. DORIGO, T. STÜTZLE (2004). *Ant Colony Optimization*. MIT Press, Cambridge.

M. DRUGAN, D. THIERENS (2004). *Evolutionary Markov chain Monte Carlo*. In P. LIARDET (ed.), *Proc. Sixth Intern. Conf. on Artificial Evolution - EA 2003*. Springer, Berlin, pp. 63–76.

D. B. FOGEL (1998). *Evolutionary computation: toward a new philosophy of machine intelligence*. IEEE Press, New York.

L. J. FOGEL, A. J. OWENS, M. J. WALSH (1966). *Artificial intelligence through simulated evolution*. Wiley, New York.

M. FORLIN, I. POLI, D. DE MARCH, N. PACKARD, G. GAZZOLA, R. SERRA (2008). *Evolutionary experiments for self-assembling amphiphilic systems*. Chemometrics and Intelligent Laboratory Systems, 90, pp. 153–160.

C. GAETAN (2000). *Subset arma model identification using genetic algorithms*. Journal of Time Series Analysis, 21, pp. 559–570.

C. J. GEYER (1991). *Markov chain Monte Carlo maximum likelihood*. In E. M. KERAMIDAS (ed.), *Computing Science and Statistics, Proc. 23rd Symposium on the Interface*. Interface Foundation, Fairfax Station, pp. 156–163.

W. R. GILKS, R. RICHARDSON, D. J. SPIEGELHALTER (1996). *Markov Chain Monte Carlo in Practice*. Chapman and Hall/CRC.

G. GOSWAMI, J. S. LIU (2007). *On learning strategies for evolutionary Monte Carlo*. Statistics and Computing, 17, pp. 23–38.

J. HANDL, J. KNOWLES, M. DORIGO (2003). *Ant-based clustering: a comparative study of its relative performance with respect to k-means, average link and 1 d-som*. Technical Report TR/IRIDIA/2003-24, IRIDIA, Université Libre de Bruxelles, Belgium. Http://wwwcip.informatik.uni-erlangen.de/šijuhand/TR-IRIDIA-2003-24.pdf.

C. C. HOLMES, B. K. MALLICK (1998). *Parallel Markov chain Monte Carlo sampling*. mimeo, Dept. of Mathematics, Imperial College, London.

G. KAPETANIOS (2007). *Variable selection in regression models using nonstandard optimisation of information criteria*. Computational Statistics & Data Analysis, 52, pp. 4–15.

J. KENNEDY, R. EBERHART (1995). *Particle swarm optimization*. In *Proc. IEEE Conference on Neural Networks*. Piscataway, pp. 1942–48.

J. KENNEDY, R. EBERHART (2001). *Swarm Intelligence*. Morgan Kaufmann, San Mateo.

H.-J. KIM, K.-S. SHIN (2007). *A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets*. Applied Soft Computing, 7, pp. 569–576.

J. R. KOZA (1992). *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge.

P. LARRAÑAGA, J. A. LOZANO (2002). *Estimation of distribution algorithms: a new tool for evolutionary optimization*. Kluwer, Boston.

F. LIANG, W. H. WONG (2000). *Evolutionary Monte Carlo: applications to c_p model sampling and change point problem*. Statistica Sinica, 10, pp. 317–342.

F. LIANG, W. H. WONG (2001). *Real-parameter evolutionary Monte Carlo with applications to Bayesian mixture models*. Journal of the American Statistical Association, 96, pp. 653–666.

J. A. LOZANO, P. LARRAÑAGA, I. INZA, G. BENGOETXEA (2006). *Towards a new evolutionary computation. Advances in estimation of distribution algorithms*. Springer, Berlin.

E. MARINARI, G. PARISI (1992). *Simulated tempering: a new Monte Carlo scheme*. Europhysics Letters, 19, pp. 451–458.

U. MAULIK, S. BANDYOPADHYAY (2003). *Fuzzy partitioning using real coded variable length genetic algorithm for pixel classification*. IEEE Transactions on Geoscience and Remote Sensing, 41, pp. 1075–1081.

T. MINERVA, S. PATERLINI (2002). *Evolutionary approaches for statistical modelling*. In D. B. FOGEL, M. A. EL-SHARKAM, G. YAO, H. GREENWOOD, P. IBA, P. MARROW, M. SHAKLETON (eds.), *Evolutionary Computation 2002. Proceedings of the 2002 Congress on Evolutionary Computation*. IEEE Press, Piscataway, vol. 2, pp. 2023–2028.

C. A. MURTHY, N. CHOWDHURY (1996). *In search of optimal clusters using genetic algorithms*. Pattern Recognition Letters, 17, pp. 825–832.

C. S. ONG, J. J. HUANG, G. H. TZENG (2005). *Model identification of ARIMA family using genetic algorithms*. Appl. Math. Comput., 164, pp. 885–912.

K. V. PRICE, R. STORN, J. LAMPINEN (2005). *Differential evolution, a practical approach to global optimization*. Springer, Berlin.

C. R. REEVES, J. E. ROWE (2003). *Genetic algorithms - Principles and Perspective: A Guide to GA Theory*. Kluwer, London.

G. O. ROBERTS, W. R. GILKS (1994). *Convergence of adaptive direction sampling*. Journal of Multivariate Analysis, 49, pp. 287–294.

A. ROVERATO, I. POLI (1998). *A genetic algorithm for graphical model selection*. Journal of the Italian Statistical Society, 7, pp. 197–208.

R. SABATIER, C. REYNÉS (2008). *Extensions of simple component analysis and simple linear discriminant analysis using genetic algorithms*. Computational Statistics & Data Analysis, 52, pp. 4779–4789.

T. STÜTZLE, H. H. HOOS (2000). *Max–min ant systems*. Future Generation Computer Systems, 16, pp. 889–914.

L. Y. TSENG, S. B. YANG (2001). *A genetic approach to the automatic clustering problem*. Pattern Recognition, 34, pp. 415–424.

D. H. WOLPERT, W. G. MACREADY (1997). *No free lunch theorems for optimization*. IEEE Transactions on Evolutionary Computation., 1, pp. 67–82.

B. WU, C.-L. CHANG (2002). *Using genetic algorithms to parameters (d,r) estimation for threshold autoregressive models*. Computational Statistics & Data Analysis, 38, pp. 315–330.

SUMMARY

Evolutionary Computation Methods and their applications in Statistics

A brief discussion of the genesis of evolutionary computation methods, their relationship to artificial intelligence, and the contribution of genetics and Darwin's theory of natural evolution is provided. Then, the main evolutionary computation methods are illustrated: evolution strategies, genetic algorithms, estimation of distribution algorithms, differential evolution, and a brief description of some evolutionary behavior methods such as ant colony and particle swarm optimization. We also discuss the role of the genetic algorithm for multivariate probability distribution random generation, rather than as a function optimizer. Finally, some relevant applications of genetic algorithm to statistical problems are reviewed: selection of variables in regression, time series model building, outlier identification, cluster analysis, design of experiments.