A Survey of Genetic Algorithms

M. Tomassini
Centro Svizzero di Calcolo Scientifico, Manno
and
Laboratoire de Systèmes Logiques
Ecole Polytechnique Fédérale de Lausanne
Switzerland

To be published in Volume III of Annual Reviews of Computational Physics, World Scientific

Abstract

Evolutionary algorithms are an important emergent computing methodology. They have aroused intense interest in the past few years because of their versatility in solving difficult problems in the optimization and machine learning fields. Many applications to several different areas have been reported and the field is still in expansion. We will first briefly review the history and the methodological basis of evolutionary algorithms, followed by a simple example of their functioning. Parallel evolutionary algorithms will then be introduced, showing their good match to today's parallel and distributed computers. We will then look at a couple of applications and, finally, references and comments to bibliographic and other information on evolutionary methods will be given to allow readers to broaden their knowledge in the field.

1 Introduction

Evolutionary Algorithms (EAs) are a hot topic these days. Although they are probably a fashionable theme, there is also much solid work being done and the steady adoption of evolutionary computing methodologies not only in research but also for industrial and commercial problem-solving activities is a sure sign that the approach is sound and competitive. EAs are here to stay then and we will try to find out how they work and why they offer good solutions for difficult problems.

Evolutionary Algorithms are search and optimization procedures that find their origin and inspiration in the biological world. The Darwinian theory of evolution, with the survival of the fittest in a changing environment seems to be generally accepted, at least on grounds of accumulated evidence so far on the earth. EAs try to abstract and mimic some of the traits of the ongoing struggle for evolution in order to do a better job in problems that require adaptation, search and optimization. However, it would be wrong to blindly identify simulated evolution with actual biological evolution. It is much better to consider ideas from the theory of evolution as being an inspiration for finding good artificial adaptive processes. After all, evolution took millions of years, is an ongoing process and operates in an exceedingly complex system of interactions. Evolution-inspired methodologies can and should only capture major distinctive features of natural evolution. Since we are in fact dealing with man-made systems, we should also feel free of using whatever device works well for a given class of problems, even if it has no direct biological counterpart, provided that some theoretical basis can be found for its use. Furthermore, artificial evolution runs at electronic speeds and is amenable to mathematical and statistical analysis.

The first part of this chapter will be devoted to a survey of the origins, motivations and developments in the field of simulated evolution. *Evolutionary Algorithms* is a general term effectively encompassing a number of related but not identical methodologies that all exploit ideas from natural evolution and selection. *Genetic Algorithms*, *Evolution Strategies* and *Evolutionary Programming* are the prominent approaches with *Genetic Programming* rapidly coming into play. Reasons of space will prevent us from describing all the models and their relationships. In the second part we will therefore concentrate on the widely used *Genetic Algorithms* (GAs), introducing them through an easy example. To put the whole subject into perspective, pointers will be given to reference work in the other evolutionary methods and to the interrelations between them. The term *Evolutionary Algorithm* will still be used in this review in general settings or whenever it does not lend itself to misunderstanding. The term *Genetic Algorithms* will be preferred for the more technical discussions.

Evolutionary computing offers many possibilities for parallel and distributed execution because many steps are independent. In fact, if the natural metaphor is to be followed, evolutionary algorithms are parallel in the first place, since evolution takes place with individuals acting simultaneously in spatially extended domains. A sequential execution setting thus appears as an unnecessary constraint. Section **??** gives an extensive discussion of parallelism in EAs.

The number of published papers on evolutionary computation has dramatically increased over the last few years. Evolutionary algorithms have been applied to many problems in diverse research and application areas such as hard function and combinatorial optimization, neural nets evolution, planning and scheduling, industrial design, management and economics, machine learning and pattern recognition. It is not my aim here to give a full account of current applications of EAs. A few representative applications will be briefly described but I shall provide a number of pointers to the relevant literature.

Because of the explosive growth of the field, there is a risk for the newcomer to get lost in the multi-faceted world of evolutionary computing. For this reason a commented bibliography is given. Although far from exhaustive, this list should prove useful for beginners. On the other hand, groups working in evolutionary computing are very enthusiastic and willing to share their results and ideas. This has given rise to mailing lists, public domain software and discussion forums. To make the life of the interested reader easier, I will provide information about how to access this sources.
Let us now turn to the fascinating history of evolutionary computing.

2 The Genesis of Genetic Algorithms

Work on what is nowadays called evolutionary computing started in the sixties both in the United States and in Europe. John Holland and his associates at the University of Michigan were interested in artificial complex systems that would be able to adapt under changing environmental conditions. The idea was that, in order for a population of individuals to collectively adapt in some environment, it should behave like a natural system where survival is promoted by the elimination of useless or harmful traits and by rewarding useful behaviour. Holland's insight was his abstraction into the genetic algorithm of the fundamental biological mechanisms permitting system adaptation into a form that can be expressed mathematically and simulated on a computer for a wide range of problems.

The link between an actual search and optimization problem and the GA is the individual. Each individual represents a feasible solution in some problem space through a suitable mapping. The mapping from problem space to individuals and the reverse mapping have historically been done through strings of binary digits. Introduced by Holland, bit strings are general and they allow some theoretical results about GAs to be obtained. However, bit encoding is not always the best choice and we will see in **??** that other representations are possible and have been used.

A GA is an iterative procedure which maintains a constant population size and works as follows. An initial population of a few tens to a few hundreds individuals is generated at random or heuristically. During each iteration step, called a *generation*, the individuals in the current population are evaluated and given a fitness value. To form a new population, individuals are selected with a probability proportional to their relative fitness. This ensures that the expected number of times an individual is chosen is approximately proportional to its relative performance in the population, so that good individuals have more chances of being reproduced. This selection procedure alone cannot generate any new point in the search space. GAs traditionally use two *genetic operators*: *crossover* and *mutation* for generating new individuals i.e, new search points. Crossover is the most important recombination operator: it takes two individuals called *parents* and produces two new individuals called the *offspring* by swapping parts of the parents. In its simplest form the operator works by exchanging substrings after a randomly selected crossover point. Through crossover the search is biased towards promising regions of the search space. The second operator, mutation, is essentially background noise that is introduced to prevent premature convergence to local optima by randomly sampling new points in the search space. To bit strings, mutation is applied by flipping bits at random in a string with a certain probability called the mutation rate.

GAs are stochastic iterative algorithms without converge guarantee. Termination may be triggered by reaching a maximum number of generations or by finding an acceptable solution.

The following general schema summarizes a standard genetic algorithm:

```
produce an initial population of individuals
while termination condition not met do
    evaluate the fitness of all individuals
    select fitter individuals for reproduction
    produce new individuals
    generate a new population by inserting some new good
    individuals and by discarding some old bad individuals
    mutate some individuals
end while
```

In the next section a tutorial example of a simple problem solved with a plain GA will be presented and discussed in detail. A classic source for an in-depth discussion of GAs, including the historical aspects, is Goldberg's book [1].

Let us now move for a moment to the other side of the Atlantic. While Holland was inventing GAs in the States, similar concepts were making their appearance in Germany. Ingo Rechenberg and H.-P. Schwefel wished to imitate the principles of natural evolution to achieve robust algorithms for parameter optimization problems. This approach goes under the name of *Evolution Strategies* (ESs). In its original form evolution strategies work with continuosly changing parameters represented as floating point numbers, rely on mutation as the only genetic operator and the population just had two members, the parent and the offspring. Later a multimembered population was used. Here is an outline of a typical implementation for a numerical optimization problem, where a coordinate vector corresponding to the optimum of a function is sought:

- An initial population of parent vectors $\mathbf{x}_i,\ i = 1, ..., N$ is selected at random from a uniform distribution.
- An offspring vector is obtained from each parent by adding a normally distributed random number to each vector component.
- The selection operator determines which of the vectors are to be kept for the next generation by choosing the n vectors with the best fitness among parents and offspring.
- The process of generating new vectors and evaluating the whole population continues until a satisfactory solution is found.

The above description is a simplified one. A more detailed discussion with references to the original work is to be found in Michalewicz's book [2] and in [10].

Evolutionary Programming is somewhat similar in spirit to evolution strategies in that it also uses mutation as the main genetic operator. This avenue of investigation originated in the United States in the sixties and represents problem solutions by a population of finite-state machines. Offspring machines are created by randomly mutating in various ways each parent machine. Parent and offspring are assigned a payoff and the best machines are retained to form the new population while the worst individuals die in order to maintain a constant size population.

The distinctive trait of evolution strategies and evolutionary programming with respect to genetic algorithms is that in the latter the simulated evolution takes place at the *genotypic* level, that is at the level of coding sequences, whereas the former put the emphasis on *phenotipic* adaptation i.e., the behavioural expression of a genotype in a specific environment.

With time, Evolution Strategies and Genetic Algorithms have converged somewhat with ESs introducing a form of recombination of individuals and GAs adopting the idea of floating-point coding of chromosomes for numerical work and the self-adaptation of mutation rates characteristic of ESs.

A recent account of evolutionary programming along with a comparison with genetic algorithms and evolution strategies can be found in [3].

3 A Simple Example

Summarizing what has been said in the previous section, we see that the essential ingredients of a genetic algorithm are the following:

- a constant size population of individuals, usually randomly initialized.
- each individual represents a point in the search space for a given problem through a suitable coding.
- a fitness value is assigned to each individual in the population.
- individuals are ranked and selected according to their fitness in such a way that more fit individuals are more likely to reproduce.
- genetic operators such as crossover and mutation are applied to pairs of individuals or to single individuals in order to produce new individuals i.e., new feasible solutions to a problem.

In order to explain how GAs work, I will present a simple example. GAs have been largely used in optimization and, although they are not limited to that field, their workings are probably better understood in an optimization setting. The problem is not a mathematically hard one, it could be solved by hand or with a number of other established methods and its value is purely illustrative.

The non-constrained function minimization problem can be cast as follows. Given a function $f(x)$ and a set $D \in R^n$, find x^* such that:

$$f(x^*) = \min\{f(x) \mid \forall x \in D\}$$

where $x = (x_1, x_2, \ldots, x_n)^T$. For maximization, simply replace f with $-f$. Let us consider the following function (see Fig.1):

$$f(x) = - \mid x \sin(\sqrt{\mid x \mid}) \mid$$

The problem is to find x^* in the interval $[0, 512]$ which minimizes f. Since $f(x)$ is symmetric, studying it in the positive portion of the x axis will suffice.

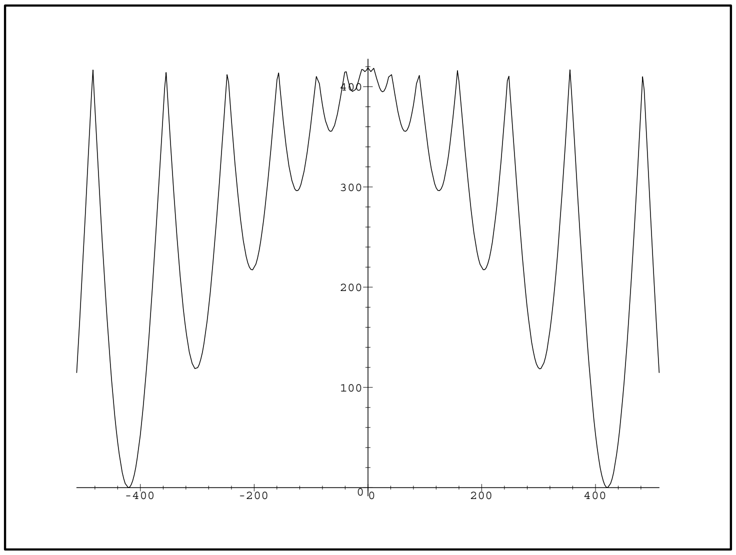

Figure 1: Graph of $f(x)$ in $[-512, 512]$.

Let us examine in turn the components of the genetic algorithm for solving the given problem.

The initial population will be formed by 50 randomly chosen trial points in the interval $[0, 512]$. Therefore, one individual is a value of the real variable x.

A binary string will be used to represent the values of x. The length of the string will be a function of the required precision, the longer the string the better the precision. For example, if each point x is represented in 10 bits then 1024 different values are available for covering the interval $[0, 512]$ with 1024 points, which gives a granularity of 0.5 for x i.e., the genetic algorithm will be able to sample points no less than 0.5 apart from each other.

The strings (0000000000) and (1111111111) will represent respectively the lower and upper bounds of the search interval. Any other 10-bit string will be mapped to an interior point. For mapping the binary string to a real number the string is first converted to a decimal number and then to the corresponding real x. Finally note that we used 10-bit strings for the sake of illustration: in real applications finer granularities and therefore longer strings are often needed.

The fitness of each sample point x is simply the value of the function at that point. Since we want to minimize f, the lower the value of $f(x)$, the fitter is x. [1]

How are strings selected for reproduction as a function of their fitness? There are several possibilities but here we will explain *fitness-proportionate* selection, one of the simplest. Alternatives to this well-known method will be briefly presented later. After having found the fitness of each individual f_i in a given generation, one first forms the sum:

$$S = \sum_{i=1}^{\texttt{popsize}} f_i,$$

which is the total population fitness. Then a probability is assigned to each string as follows:

$$p_i = \frac{f_i}{S}$$

Finally, a cumulative probability is obtained for each individual by adding up the fitnesses of the preceding population members:

$$c_i = \sum_{k=1}^{i} p_k, \quad i = 1, 2, \ldots, popsize$$

A random number r uniformly distributed in $[0, 1]$ is drawn *popsize* times and each time the i-th string is selected such that $c_{i-1} < r \leq c_i$. When $r < c_1$, the first string is selected. This process can be visualized as the spinning of a biased roulette wheel divided into *popsize* slots, each with a size proportional to the individual's fitness. For the sake of illustration, suppose that there are only four strings with the following p_i values: $p_1 = 0.30, p_2 = 0.20, p_3 = 0.40, p_4 = 0.10$. Thus we have: $c_1 = 0.30, c_2 = 0.50, c_3 = 0.90, c_4 = 1.0$. Now, imagine that a random

[1] We added a positive constant to our function definition to make it ≥ 0 in the given interval. This is required by some individual selection methods but has no influence on the general argument.

number $r = 0.25$ is drawn. Since $r < c_1$, individual 1 will be selected. If r were 0.96 then individual 4 would be selected ($c_3 < 0.96 < c_4$).

With *roulette-wheel* selection fittest members have proportionally more chances of being reproduced and strings can be selected more than once. For this method to work, the fitness values should be positive numbers since we are using probability measures.

Once the new population has been produced, strings are paired at random and recombined through crossover. Here we will explain one-point crossover. Assume that the following two strings have been selected for recombination:

$$0010011010 \quad \textit{and} \quad 1110010001$$

a position is selected at random between 1 and the length of the string minus one, each position being equally likely. Suppose that position 6 has been chosen (marked by the vertical bar):

$$001001 \mid 1010$$

$$111001 \mid 0001$$

Then, after swapping all bits from position 6 to the end of the string one obtains two new strings called the *offspring*:

$$001001 \mid 0001$$

$$111001 \mid 1010$$

These two new individuals will enter the new population in place of their parents. Crossover is applied with a certain frequency called crossover rate p_c, which means that any given individual takes part in the recombination if a uniformly distributed random variable in the interval $[0, 1]$ has a value $\leq p_c$. A common empirical value for p_c is 0.6.

After crossover, mutation can be applied to population members with a frequency p_m around 0.01. The usual interpretation of bit mutation rate is the following: for each string in the population and for each bit within the string generate a random number r between 0 and 1, if $r \leq p_m$ flip the bit. These values of p_c and p_m have been arrived at by repeated experimentation and trial and error and have nothing sacred in themselves. In more sophisticated GAs crossover and especially mutation rates do not need to stay constant during a run.

What is the role of these genetic operators? There is an abundant literature about different variants of crossover and mutation and their relative importance. In the classical GA view crossover is the fundamental operator and mutation only plays an ancillary role. In this view, the importance of crossover comes from the fact that it is believed to combine beneficial traits of both parents, thereby increasing the likelihood of generating fitter individuals, whereas mutation can only affect one individual at a time. This has to do with the apparent usefulness of sexual reproduction in nature and its general diffusion, in spite of the fact that it requires finding a mate and can make individuals more vulnerable to predators during the search. In a more symbolical vein, the usefulness of crossover seems to be related to the combination of so-called *building blocks* i.e. better than average substrings coming from different individuals (see the next section). Trough crossover we thus try combinations of strings that have already been proved to be relatively good. Mutation is still needed because even if selection and crossover together search new solutions, they tend to cause rapid convergence and there is the danger of loosing potentially useful genetic material. We shouldn't forget that we are in fact restricted to relatively small sample sizes in practice, whence the possibility of sampling errors. In order to reintroduce diversity and to avoid search stagnation, bit mutations are allowed as described above. However, mutation frequencies have to be low, otherwise the search tends to degenerate into a random walk.

The relative importance of mutation and crossover is still controversial and some evolutionary techniques such as evolution strategies and evolutionary programming have selection and more sophisticated versions of mutation, and not crossover, as their principal evolutionary operators (see for instance refs.[10] and [11]).

Equipped with these notions, let us now come back to our function minimization problem and run the GA. [2]

As measures of the quality of the search we use here the average population fitness during a generation and the best individual found. Since generation 0 is randomly initialized, it is to be expected that neither the average nor the best fitnesses are very good. In fact, in a particular run we found the following evolution:

Generation	Best	Average
0	1.0430	268.70
3	1.0430	78.61
9	0.00179	32.71
18	0.00179	14.32
26	0.00179	5.83
36	0.00179	2.72
50	0.00179	1.77
69	0.00179	0.15

From the table, one sees that, starting from a random population, there is a fairly rapid improvement in the first generations. The minimum is already found at generation 9. However, the population average fitness continues to improve until the population becomes little differentiated and the fitness levels-off. This is a quite general behaviour of all evolutionary algorithms. In this simple problem there is practically no risk for the algorithm to get trapped in a local minimum. In harder problems, a compromise must be reached between exploitation of good regions, i.e. local improvement, and further exploration of the problem space, to avoid missing better extrema as far as possible.

At the end of the run (generation 70), the five best solutions found where those shown in the following table (where the *Generation* column says at which generation a given solution has been found).

[2]All the runs described in this section have been done with the *GENESIS* program [12]. The tables shown have been edited to improve readability.

x	$f(x)$	Generation
421.5	0.04832	8
422.0	0.15785	6
421.0	0.00179	9
420.5	0.01824	8
420.0	0.09763	12

All points are clustered around the absolute minimum ($x = 421.0, f(x) = 0.001794$) which has thus been found within the given resolution limits. One last remark is in order. Genetic algorithms are stochastic, thus their performance varies from run to run (unless the same random number generator with the same seed is used). Because of this, the average performance over several runs is a more useful indicator of their behaviour than a single run.

This was an easy problem to solve for GAs as well as for any other method. However, GAs have been shown to be effective for hard mathematical optimization of multimodal functions with tens of variables [9].

4 Why GAs work: Schemata and Building Blocks

In this section we will look in a little greater detail into the standard genetic algorithm workings in order to see why GAs constitute an effective search procedure. Remaining in the realm of binary string representation of individuals, let us consider the symbol alphabet $\{0, 1, \#\}$ where $\{\#\}$ is a special *wild card* symbol that matches both 0 and 1. A *schema* is a string with fixed and variable symbols. For example, the schema [01#1#] is a template that matches the following strings: [01010], [01011], [01110] and [01111]. The symbol # is never actually manipulated by the GA: it is only a notational device that makes it easier to talk about *families* of strings.

Holland's idea was that every evaluated string actually gives partial information about the fitness of the set of possibles schematas of which the string is a member. This is a manifestation of what he called *implicit parallelism*, not to be confused with the kind of parallelism to be discussed in section **??**. Then he analyzed the influence of reproduction, crossover and mutation on the expected number of schemata when going from one generation to the next. The details of the analysis are relatively simple but cannot be reported here. A good discussion can be found in ref. [1]. We will only outline the main results and their significance.

Under fitness-proportionate replication, the number m of individuals in the population belonging to a particular schemata H at time $t+1$ is related to the same number at time t by:

$$m(H, t+1) = m(H, t)(f_H(t)/(\overline{f}(t))$$

where $f_H(t)$ is the average fitness value of the strings representing schema H, while $\overline{f}(t)$ is the average fitness value over all strings in the population. If one assumes that a particular schema remains above the average by a fixed amount $c\overline{f}(t)$ for a number t of generations then the solution of the above recurrence is the following exponential growth equation:

$$m(H, t) = m(H, 0)(1+c)^t$$

Where $m(H, 0)$ stands for the number of schemata H in the population at time 0, c is a positive constant and $t \geq 0$. The significance of this result is that fitness-proportionate reproduction allocates exponentially increasing number of trials to above-average schemata.

Now crossover and mutation enter into the picture. The effect of crossover, which breaks strings apart, is to diminish the exponential increase by a quantity that is proportional to the crossover rate p_c and depends on the *defining length* δ of a schema and on the string length l:

$$p_c \frac{\delta(H)}{l-1}$$

The defining length δ of a given schema is the distance between the first and the last fixed string positions. For example, for the schema [01#1#] $\delta = 4 - 1 = 3$ and for [##1#1010] $\delta = 8 - 3 = 5$. Intuitively, one sees that short defining length schemata will be less disrupted by single-point crossover. The result is that above-average schemata with short defining lengths will still be sampled at an exponentially increasing rate. This above-average, short defining length schemata are the so-called *building blocks* and play an important role in the theory.

The effects of mutation are straightforward to describe. If the bit mutation probability is p_m, then the probability of survival of a single bit is $1-p_m$. Since single bit mutations are independent, the total survival probability is thus $(1-p_m)^l$, where l is the string length. But since we are talking about schemata, only the fixed (i.e. non wild card) positions matter. This number is called the *order* $o(H)$ of a schema H and equals l minus the number of don't care symbols. For example, the two schemata above have $o = 3$ and $o = 5$ respectively. Then the probability of surviving a mutation for a schema H is $(1-p_m)^{o(H)}$ which, for $p_m \ll 1$ can be approximated by $1 - o(H)p_m$.

Putting together the effects of reproduction, crossover and mutation, we are led to Holland's so-called *schema theorem*:

$$m(H, t+1) \geq m(H, t)\frac{f_H(t)}{\bar{f}(t)}[1 - p_c\frac{\delta(H)}{l-1} - o(H)p_m]$$

This result essentially says that the number of short, low-order, above-average schemata grows exponentially in subsequent generations of a genetic algorithm.

Although the schema theorem is an important result, it was obtained under somewhat idealized conditions. Both the individual representation and the genetic operators can be different from those used by Holland. The building-block hypothesis has been found reliable in many cases but it also depends on representation and genetic operators and it is easy to find or to construct problems for which it is not verified. These so-called *deceptive* problems are being studied since a few years in order to find out what are the inherent limitations of genetic algorithms and which representations and operators, if any, can make them more easily tractable. In spite of the above limitations, the theory sketched in this section represents a firm footing for the workings of standard genetic algorithms. Refs. [1], [13] and [14] go into much more detail.

5 More Advanced Topics and Extensions

In this section we hint at several extensions and variations that somewhat complicate the rather neat image of a GA given earlier. This will bring us closer to the way GAs are actually used by practitioners and will also partly explain their generality and flexibility. We will look first at coding issues, including non-binary and non-fixed-length representations. Then alternatives to fitness-proportionate selection will be introduced in section **??**. The following section **??** describes some different forms of the genetic operators, especially crossover. Finally, we introduce hybrid algorithms, in which problem domain knowledge is brought into the GA in various ways making them less general but often much more efficient. Although we will touch upon the more important issues in the following sections, we cannot do justice to the amount of research that has been done or is being done. However, the interested reader will find references to more detailed presentations.

5.1 Representation

Binary coding has been the usual individual representation in genetic algorithms for a long time. Binary strings are sufficiently general but they are not always the more natural or the more adequate representation. Consider, for instance, numerical parameter optimization problems such as the example in section **??**. There, the precision was a function of the number of bits in the bit string representing an individual point. To attain a sufficient precision requires many bits and the problem is all the more serious if one wants to tackle multidimensional problems. Dealing with very long bit strings is time-consuming and the search spaces are enormous. Therefore, wouldn't it be better to consider the more natural floating point representation in these cases? This has indeed been done with very good results ([2], [9]). Obviously, switching to floating point representation requires careful rethinking of the genetic operators, which are going to be different from those used for bit strings.

Representation issues also appear when dealing with *combinatorial* optimization problems i.e., those discrete variable problems in which a particular solution out of a finite set of feasible solutions is sought [15]. For example, the *shortest path* problem on a directed graph is an *easy* combinatorial problem, meaning that it takes time proportional to a polynomial function of the instance size to solve in the worst case. However there are many important combinatorial optimization problems that are intrinsically *hard* i.e., their time complexity is exponential. Typical representatives of this class are the *Traveling Salesman Problem* (TSP) and the *Hamilton Circuit* problem. In TSP we are to find the shortest tour that visits each node of a complete weighted graph G exactly once. In the Hamilton circuit problem the question is: given a graph G, is there a circuit in G visiting all nodes exactly once?

There are no known efficient algorithms for hard combinatorial problems like the TSP and Hamilton circuit. Furthermore, these problems are paradigmatic versions of very important management problems in the fields of sequencing, routing and scheduling. Therefore, it is important to be able to quickly find good solutions to large instances of these problems, even if the solution is not globally optimal. Approximation and heuristic algorithms of various kinds, including genetic algorithms, have been found effective.

When using GAs for this kind of problems, representation issues surface again. For graphs, a natural representation of an individual is an integer vector, where the integers represent some ordering of the nodes, instead of a binary string. If we take again TSP as an example, solutions may develop by crossover and mutation that are not legal tours, not only with a binary representation, but also when using integers. These new illegal individuals need to be repaired or penalized in some way. Another possibility consists in defining representations and genetic operators in such a way that only legal solution can be produced. At any rate, the new representation and operators will bear little resemblance to the classical binary string based ones. Furthermore, theoretical results obtained for bit strings are not immediately transferable to other representations. On the other hand, an abundance of accumulated circumstantial and empirical evidence, tend to suggest that specially developed genetic representations and operators may lead to efficient evolutionary solutions to difficult problems.

Ref. [2] contains a discussion of genetic representation issues for combinatorial and numerical problems including constrained ones, together with references to original work.

A completely different representation is suggested by a new way of using evolutionary algorithms called *genetic programming*, fully described in ref. [17]. Genetic programming is a major variation of genetic algorithms in which the evolving individuals are themselves computer programs instead of fixed length strings from a rather limited alphabet of symbols. Programs are represented as trees with ordered branches in which the internal nodes are functions and the leaves are the so-called terminals of the problem. The search space in genetic programming is the space of all computer programs composed of functions and terminals appropriate to the problem domain.

Suitable functions and terminals are determined for the problem at hand and an initial random population of trees (programs) is constructed. From there on the population evolves in the usual GA way with fitness being associated to the actual execution of the program (individual) and with genetic operators adapted to the tree representation. The crossover operation starts by selecting a random crossover point in each parent tree and then exchanges the sub-trees, giving rise to two offspring trees. Mutation is implemented by randomly removing a subtree at a selected point and replacing it with a randomly generated subtree. There are also provisions for preventing trees from becoming too deep, for simplifying trees and for compressing trees that perform a useful functions into a single reusable module.

Genetic programming has been shown to be able to automatically breed programs able to solve, or approximately solve, a variety of relatively simple problems from many fields [17]. It remains to be seen whether the methodology can be extended to automatically evolve problems for more difficult tasks and for general programming.

5.2 Selection

In section **??** fitness-proportionate selection was introduced. This selection method is not without problems however. One problem is that, after a while, since better individuals get more copies in successive generations, the differences in fitness between individuals become small which renders selection ineffective. In this case the *selection pressure* need to be augmented to allow the better individuals to reproduce more often than they would under the normal fitness evaluation.

Another problem is the possible existence of a *super individual* in the population i.e., an individual with an unusually high fitness. With fitness-proportionate reproduction this individual will get many copies in successive generations and rapidly come to dominate the population, thus causing premature convergence to a possibly local optimum.

It is possible to partially avoid these effects by suitably *scaling* the evaluation function, which amounts to the use of a modified fitness measure. Several scaling methods have been suggested and are discussed for example in [1].

Another approach to mitigate the above effects is to use selection methods that do not allocate trials proportionally to fitness. Two such methods are *ranking selection* and *tournament selection*.

In ranking selection, the individuals in the population are ordered by fitness and copies assigned in such a way that the best individual receives a predetermined multiple of the number of copies than the worst one. Rank selection reduces the dominating effects of super individuals without need for scaling and, at the same time, it exacerbates the difference between close fitness values, thus increasing the selection pressure in stagnant populations. Ranking selection methods have been used with some success, on the other hand, they ignore the information about relative fitness of different individuals and violate the schema theorem.

In tournament selection a number n of individuals is selected at random with uniform probability and the best one among them finds its way into the new population. The winner can also be chosen probabilistically. The process is then repeated *popsize* times. The selection pressure is proportional to the *tournament size* n. A widely used value of n is two. Tournament selection has the advantage that it need not be global so that local tournaments can be held simultaneously in a spatially organized population (see section **??**).

A quite satisfactory treatment of selection methods together with many references is to be found in ref. [2].

5.3 Genetic Operators

The one-point crossover and mutation used up to now are the original versions of the genetic operators. Simple and inspired by biology they have nevertheless some drawbacks in practice. Consequently, many variants have been proposed. Let us start with crossover. One natural extension of the one-point crossover is the *multi-point crossover*. For instance, in two-points crossover there are two cut points (marked by the vertical bars) and substrings are swapped between the two points:

001 | 101 | 1010

111 | 001 | 0001

001 | 001 | 1010

111 | 101 | 0001

According to some researchers, multi-point crossover is more apt to combine certain good features present in strings.

Another widely used crossover type is *uniform crossover*. Given two parent strings, for each bit in the first offspring a bit in the corresponding position is copied randomly with some probability from one of the parents. The second offspring gets the corresponding bit from the remaining parent. For example, given the two parents above and a probability of 1/2, suppose that the following series of random choices is made (where 1 stands for the first parent and 2 for the second):

1221211212

then we would obtain the following offspring:

0111011011

1010010000

Uniform crossover violates the customary form of the schema theorem and is less likely to preserve good building blocks. However, for some problems, it has given good results. For a good discussion of crossover-related issues and further references, see chapter 4 of ref. [2].

Mutation has been less studied than crossover in the GA literature. Worth of note are adaptive mutation schemes, partly borrowed from evolution strategies, in which either the rate or the form of mutation or both, vary during a GA run. For instance mutation is sometimes defined in such a way that the search space is explored uniformly at first and more locally towards the end, in order to do a kind of local improvement of candidate solutions. Again, further information on sophisticated mutation techniques can be found in ref. [2].

5.4 Hybrid Algorithms

Genetic algorithms are a robust, general-purpose search procedure. They can quickly explore huge search spaces and find those regions that have above-average fitness. However, when it comes to actually finding global optima, they sometimes run into difficulties because they lack focus in the search. This in turn raises the question as to what extent GAs can be competitive for real-world applications when compared to more specialized algorithms and heuristics. The answer may lie in *hybrid genetic algorithms*. Hybrid genetic algorithms work by incorporating a fast and efficient problem-specific search procedure. They also tend to use encodings and genetic operators that are tailored to the problem to be solved. By doing so, very efficient algorithms can be produced, as demonstrated by some recent work ([18],[19]). Hybrid GAs are even less amenable to theoretical analysis than standard genetic algorithms but they are very interesting in practice and their use is increasing. A readable description of the motivations behind hybrid GAs appears in ref. [16].

6 Parallel Evolutionary Algorithms

Parallel computing is becoming an important part of scientific computing in general since it holds the promise of improving performance by just adding processors, memory and an interconnection and putting them to work together on a given problem. By sharing the workload, it is hoped that an N-processor system will do the job nearly N times faster than a uniprocessor system, thereby allowing researchers to treat larger and more interesting problem instances. In reality, things are not so simple since several overhead factors contribute to significantly lower the theoretical performance improvement expectations. In any event, there exist many important problems that are sufficiently regular in their space and time dimensions to be suitable for parallel computing. Fortunately, evolutionary algorithms belong to this class of 'easy' parallel problems.

The original formulation of GAs by Holland and others in the seventies was a sequential one. This approach made it easier to reason about mathematical properties of the algorithms and was justified at the time by the lack of adequate software and hardware. This is no longer the case today and parallel evolutionary algorithms are becoming more common.

There are two main reasons for parallelizing an evolutionary algorithm: one is to achieve time savings by distributing the computational effort and the second is to benefit from a parallel setting from the algorithmic point of view, in analogy with the natural parallel evolution of spatially distributed populations.

We will start by describing simple though very useful parallel evolutionary algorithms whereby interesting performance improvements can be obtained without changing the general sequential evolutionary algorithm schema. In many real-world problems, the calculation of the individual's fitness is by far the most time consuming step of the algorithm. In this case an obvious approach consists in evaluating each individual fitness simultaneously on a different processor. If there are more individuals than processors, which is often the case, then the individuals to evaluate are divided as evenly as possible among the available processors. It is assumed that fitness evaluation takes about the same time for any individual. The other parts of the algorithms are as before and remain centralized. The following is an informal description of the algorithm:

```
produce an initial population of individuals
```

```
while termination condition not met do
    do in parallel
        evaluate the fitness of all individuals
    end parallel do
    select fitter individuals for reproduction
    produce new individuals
    generate a new population by inserting some new good
    individuals and by discarding some old bad individuals
    mutate some individuals
end while
```

Another method consists in running simultaneously and independently N copies of the algorithm on the N available processors. The best of the multiple independent runs is then the required result. Since EAs are stochastic, several runs are in general needed anyway to draw statistically significant conclusions. The copies must differ in the generation of the initial population and possibly in the setting of some parameters such as the crossover and mutation rate. Each processor computes for a given number of generations. In practice no communication is needed between processors except when it is desired to stop the computation when one processor has satisfactorily solved the problem before the allowed maximum number of generations.

We now turn to more genuinely parallel approaches for evolutionary algorithms. All these find their inspiration in the observation that natural populations tend to possess a *spatial* structure. As a result, so-called *demes* make their appearance. Demes are semi-independent groups of individuals or subpopulations having only a loose coupling to other neighbouring demes. This coupling takes the form of the slow migration or diffusion of some individuals from one deme to another. A number of models based on spatial structure have been proposed. The two most important categories are the *island* and the *grid* models.

The *island* model [4,5] features geographically separated subpopulations of relatively large size. Subpopulations may exchange information by allowing some individuals to migrate from one subpopulation to another with a given frequency and according to various patterns. The main reason behind this model is to periodically reinject diversity into otherwise converging subpopulations. When the migration takes place between nearest neighbour subpopulations the model is called *stepping stone*. Within each subpopulation a standard sequential evolutionary algorithm is executed between migration phases. The following is a high-level algorithmic description of the process:

```
initialize P subpopulations of size N each
generation number := 1
while termination condition not met do
  for each subpopulation do in parallel
      evaluate and select individuals by fitness
      if generation number mod frequency = 0 then
        send K<N best individuals to
        a neighbouring subpopulation
        receive K individuals from a
        neighbouring population
        replace K individuals in
        the subpopulation
      end if
      produce new individuals
      mutate individuals
  end parallel do
  generation number := generation number + 1
```

```
end while
```

Here *frequency* is the number of generations before an exchange takes place. Several individual replacement policies have been described in the literature. One of the most common is for the migrating `K` individuals to displace the `K` worst individuals in the subpopulation. It is to be noted that the subpopulation size, the frequency of exchange, the number of individuals to migrate and the migration topology are all new parameters of the algorithm that have to be set in some way. At present there is no rigorous way for choosing them. However, several empirical investigations have arrived at rather similar conclusions [4,7].

In the *grid* or *fine-grained* model [6] individuals are placed on a large toroidal two-dimensional grid, one individual per grid location. Fitness evaluation is done simultaneoulsy for all individuals and selection, reproduction and mating take place locally within a small neighborhood. With time, semi-isolated niches of genetically homogeneous individuals emerge across the grid as a result of slow individual diffusion. This phenomenon is called *isolation by distance* and is due to the fact that the probability of interaction of two individuals is a fast-decaying function of their distance. The following is a pseudo-code description of a grid evolutionary algorithm.

```
for each grid point do in parallel
  generate a random individual
end parallel do
while termination condition not met do
```

```
        for each grid point $k$ do in parallel
            evaluate individual in $k$
            select a neighbouring individual $q$
            produce offspring from $k$ and $q$
            assign one of the offspring to $k$
            mutate $k$ with probability $p_m$
        end parallel do
    end while
```

In the preceding description the neighborhood is generally formed by the four or eight nearest neighbors of a given grid point. The selection of an individual in the neighborhood for mating with the central individual can be done in various ways. Tournament selection is one of the more common and easier. Likewise, the replacement of the original individual can be done in several ways. For example, it can be replaced by the best among itself and the offspring or one of the offspring can replace it at random. The model can be made more dynamical by adding provisions for longer range individual movement through random walks, instead of having individuals interacting exclusively with their nearest neighbours [8].

Though both island and grid models can be implemented on serial machinery and constitute in this case useful variants of the standard globally communicating population genetic algorithm, they are ideally suited for parallel computers. From an implementation point of view, coarse-grained island models, where the ratio of computation to communication is high, are more adapted to multiprocessor systems and even for clusters of workstations [4,5,7]. Grid models are well adapted for massively parallel SIMD (Single Instruction Multiple Data) machines such as the Connection Machine CM-200 [8], since the necessary local communication operations, though frequent, are very efficiently implemented in hardware. Further, it should be noted that hybrid models are also possible. For example, one might consider an island model in which each island is structured as a grid of individuals interacting locally.

In general, it has been found that parallel evolutionary algorithms, apart from being significantly faster, help in relieving the premature convergence problem and are effective for multimodal optimization [4,5]. This is due to the larger total populations and to the relative isolation of the spatial regions where solutions start to co-evolve. Both of these factors help to preserve diversity while at the same time promoting local search. As in the sequential case, the effectiveness of the search can be improved at the expense of generality by permitting hill-climbing, i.e. local improvement around promising search points [9].

Until now only the space dimension entered into the picture. If we take time into account as well, then parallel evolutionary algorithms can be loosely synchronous, synchronous or asynchronous. Island models are in general loosely synchronous. They use an SPMD (Single Program Multiple Data) coarse-grain parallelism in which communication phases synchronize processes. This is not necessary and experiments have been done with asynchronous EAs in which subpopulations exchange individuals only when some internally measured level of convergence has been attained. This avoids constraining all coevolving populations to do the swaps at the same time irrespective of subpopulation evolution. This approach could well be more effective but requires the handling of a list of exchange requests and needs a common measure of population diversity. Fine-grained parallel EAs are fully synchronous when they are implemented on SIMD machines and are an example of data-parallelism.

Triggered by the recent availability of parallel computers and workstation clusters, parallel and distributed EAs have been used with success for some time. They are simple to implement and offer advantages over the straight sequential algorithm. However, parallelism introduces new degrees of freedom that have to be dealt with by the implementer, their theoretical analysis is more difficult and very little is known about their properties.

I conclude this section with a remark concerning the possibility of obtaining ‘superlinear speedup’ i.e., getting more than n-fold acceleration with n processors, when using parallel evolutionary algorithms. Although strictly speaking superlinear speedup does not arise in deterministic situations, it becomes possible when an element of chance in choice ordering is present. For example, this has been shown to be the case in tree and graph searching problems where, depending on the position of the solution and the way the search space is subdivided, parallel search may be more than n-fold effective. The problem lies in determining what are the input distributions that make the phenomenon possible. Superlinear speedup has been reported in recent work for parallel evolutionary algorithms and the same effect has been observed for stochastic algorithms of the Monte Carlo type. Many people, including the author, have observed that almost always a lower number of fitness evaluations is needed to reach the same quality of solution in the parallel than in the sequential case. This may loosely be attributed to a kind of cooperative effect in the search that is lost when working with globally communicating populations.

7 Applications of Evolutionary Algorithms

Literally hundreds of papers have been published in the last few years on EAs applications that range from industrial optimization and design ([20],[21]), neural network design ([22]), management and finance ([23]), artificial life ([24]), communication networks ([25]) and many, many others. A complete bibliography is given in ref.[26]. Clearly, it would be impossible to give here a faithful account of current applications of EAs. To give the reader a flavour of this blossoming activity, I will briefly describe two studies: one on cellular automata, which should appeal to the computational physicist and the second, to management, more typical of applied GA research.

7.1 Evolving Cellular Automata

Artificial evolutionary processes can be helpful for discovering emergent information processing capabilities in decentralized, spatially-extended models. One of the simplest model of this kind is a one-dimensional cellular automata (CA) [27].

In ref. [28] Crutchfield and coworkers described a simple computational task for a finite, one-dimensional two-state CA which is to decide whether or not a given initial CA configuration of zeroes and ones contains more than half 1s. The task is trivial for systems with a centralized control but it is difficult for a CA, in which only finite radius information transmission can take place at any given moment. The seven neighbours cellular automata rule of Gacs, Kurdymov and Levin rule (GKL) performs this task correctly for a substantial part of many randomly generated initial configurations.

The authors carried out a set of experiments in which GAs are used to evolve CA rules for the above described computational task. One-dimensional CA rules can be easily encoded as binary strings by just successively recording in the string the next states (binary) corresponding to all the neighborhood states combinations in a given rule listed in a fixed order. For example, the following rule is one of the possible 256 rules with two states and three neighbours and it is represented by the string (01011010).

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

With seven neighbours and two possible states, one has rules of length $2^7 = 128$ and the number of possible rules is huge: 2^{128}. Starting with a population of random CA rules, the authors have used as a fitness measure for a rule the number of correct classifications after a given number of CA steps over 100 initial random configurations chosen with uniform probability. As usual, the strings (rules) that performed better were selected to survive and randomly paired to produce new rules by crossover, the offspring being subject to a small mutation rate.

Computational capabilities and general patterns of rule strategies were found to automatically emerge from the simulated evolutionary process although in no case the GA-evolved pattern classification strategies were superior to the GKL rule. However, some evolved rules had remarkably good performance, close to that of the GKL rule which, for a system of this complexity, is a good result. Further findings belonging to the theory of discrete non-linear dynamical systems are discussed in [28] and related papers.

7.2 Portfolio Selection with Distributed GAs

The central problem in portfolio selection is to find a number of assets and their weights in such a way that a certain measure of risk is minimized for any given level of expected return on investment. Classically, risk is measured as the standard deviation or the variance of the probability distribution of future returns. In this framework, quadratic programming is used for solving the portfolio selection problem. More recent approaches are based on semivariance and downside risk, which roughly means that investors only perceive risk below the mean of the distribution of returns. For these new models deterministic algorithms such as quadratic programming are not very useful. There are tens or even hundreds of assets in a given portfolio and the risk-return surface is no longer convex, as when variance is used, but it becomes a very rugged, non-convex, highly multimodal landscape. When deterministic algorithms for optimization fall short, stochastic and heuristic methods may become attractive. Genetic algorithms where thus used to solve the portfolio allocation problem in ref. [29].

Choosing an optimal portfolio can be viewed as a multi-objective optimization problem in that an investor wants to minimize risk while maximising expected return. As the level of risk increases, the expected return attached to optimal portfolios draws a convex non-decreasing curve, which is called efficient frontier, which is the set of Pareto-optimal, i.e. non-dominated, portfolios. In other words, on the efficient frontier, a larger expected return corresponds to a greater risk. This can be expressed as a two-objective optimization problem in the parameters region $\mathbf{w}$:

$$\min_{\mathbf{w}}\{\mathrm{Risk}(\mathbf{w})\}$$

$$\max_{\mathbf{w}}\{\mathrm{Return}(\mathbf{w})\}$$

subject to

$$\sum_{w_i \geq 0} w_i = 1$$

These two objectives can be parametrized to yield a parametric objective

$$\min_{\mathbf{w}}\{\lambda\,\mathrm{Risk}(\mathbf{w}) - (1-\lambda)\,\mathrm{Return}(\mathbf{w})\},$$

where parameter λ is a trade-off coefficient ranging between 0 and 1. When $\lambda = 0$ the investor disregards risk and only seeks to maximize expected return; when $\lambda = 1$ the risk alone is minimized, whatever the expected return. Since there is no general way to tell which particular trade off between risk and return is to be considered the best, optimizing a portfolio means finding a whole range of optimal portfolios for all the possible values of the trade off coefficient; the investors will thus be able to choose the one they believe appropriate for their requirements. A natural way to achieve that in an evolutionary setting is to have several distinct populations evolve for a number of trade-off coefficient values. The greater this number, the finer the resolution with which the investor will be able to explore the efficient frontier. Because it is likely that slight variations in the trade-off coefficient do not significantly worsen a good solution, a natural way to sustain the evolutionary process is to allow migration or cross-breeding between individuals belonging to populations corresponding to values of the trade-off coefficient that are close together. This suggests a distributed implementation where populations are linearly arranged according to their relevant trade-off value.

Distributed GAs of the kind described in section **??** can be used to speed up the search on a cluster of workstations. The population topology used was a two-way string of processors in which exchange of individuals only takes place between nearest neighbours i.e., those with similar trade-off coefficients. Very good results have been found in [29] for portfolios with up to 150 assets. A comparison with a previous sequential solution showed that the parallel version was not only obviously faster, it also converged on the average towards better solutions in all cases over many different portfolios.

Evolutionary Computation Resources

There are several ftp and WWW sites worldwide from which useful information and public domain code can be obtained. Here we will limit ourselves to the most important ones: from there it will be easy for the interested person to follow the links to other relevant sites. The GA site 'par excellence' is *The Genetic Algorithms Archive*. This can be reached on the web at the following URL:

http://www.aic.nrl.navy.mil/galist

It can also be accessed by anonymous ftp at the following address:

ftp.aic.nrl.navy.mil in /pub/galist

This well-organized site contains a wealth of information on GA-related activities, conferences, courses and workshops, technical reports, source code and much more. It is possible to suscribe to a low-noise GA list (only digests are sent about once a week) by sending a message to: GA-list-request@aic.nrl.navy.mil.

A very complete frequently asked question (FAQ) document can be obtained either from the www site above or by anonymous ftp from:

lumpi.informatik.uni-dortmund.de in /pub/EA/docs/hhgtec-3.1.ps.gz

This FAQ, called *The Hitch-Hiker's Guide to Evolutionary Computation: A List of Frequentely Asked Questions* has been prepared by J. Heitkötter and is extremely useful. It also contains pointers and advice on techniques other than GAs, such as evolution strategies and genetic programming.

Guide to the Bibliography

Books

The classical text on Genetic Algorithms is Goldberg's book [1]. This book is eminently readable and even a bit verbose at times. It contains very good chapters on GA theory, the history of evolutionary computing and the application of evolutionary techniques in the machine learning field, an AI subject that has not been touched in the present article. On the other hand, the book is in my opinion too biased towards genetic algorithms to the detriment of other methods that are only briefly cited and it is a bit out of date, given the spectacular advances in the field since 1988. I look forward for a second edition of this excellent text.

Michalewicz's book [2] is an up-to-date and rather complete treatment of evolutionary algorithms. It gives clear explanations of the functioning of genetic algorithms but it also spends a sizeable number of pages on evolution strategies, new genetic operators and recent variants of EAs. Like Davis [17], it makes a case for adapting the genetic representation and operators to the given problem and for hybrid approaches. The chapters on combinatorial and constrained optimization are especially valuable since the original sources are scattered in many articles and conference papers. There is also a chapter on evolutionary machine learning and a good list of references. Overall, a very good book on the subject.

Handbook of Genetic Algorithms by Davis [17] is an eminently pragmatic book. The reader is taken by the hand to a whole trip about GAs in less than 100 pages. This part is very well written and easy to understand. This chapter is highly advisable to anybody wishing to rapidly grasp the subject matter. The only drawbacks are the lack of theory and a strong enphasis on GAs. But, after all, the book 'is' about genetic algorithms. The rest of the book consists in several multiauthor chapters, each one presenting a real-life application. These chapters are a bit uneven but overall they give an illuminating view of current industrial applications of GAs.
None of the above books describes parallel EAs in sufficient detail.

For genetic programming, just touched upon here, the standard reference is Koza's book [17]. Another useful reference is [30].

Evolution strategies and evolutionary programming are less well represented, both in the present article for technical reasons, and in the general literature. Apart from ref. [2], ES are well described in [10].

A recent book by D. Fogel [31] is the only one available on evolutionary programming, apart from the historical *Artificial Intelligence through Simulated Evolution* by L.J. Fogel, A.J. Owens and M.J. Walsh, published in 1964. The book is readable and more biology-oriented than other works. The underlying theme is that intelligence in the biological sense is an evolutionary property and it makes a case for phenotypic adaptation as opposed to the genotypic evolution typical of the GA.There is a good chapter on theory and some simple control and game applications are discussed at length.

Conferences and Journals

The most important journal on evolutionary algorithms is *Evolutionary Computation*, published quarterly by MIT Press. Some related journals are *Adaptive Behaviour* and *Artificial Life*, both published quarterly by MIT Press and *BioSystems*, by Elsevier. Many articles on applications are published in other journals, Alander's bibliography [26] being the more complete source to date.

Many conferences have sections on evolutionary computing. The most important dedicated conferences are the following:

- *ICGA: International Conference on Genetic Algorithms.* It takes place in the U.S. on odd-numbered years. Proceedings published by Morgan Kaufmann.
- *PPNS: Parallel Problem Solving from Nature.* International conference held in Europe in even-numbered years. Proceedings published by Springer-Verlag in the Lecture Notes in Computer Science series.
- *ICANNGA: International Conference on Artificial Neural Nets and Genetic Algorithms.* Takes place in Europe every two years. Proceedings published by Springer-Verlag.
- *Alife: International Conference on Artificial Life.* Held in the U.S. Many articles of interest for artificial evolution. Proceedings by Addison-Wesley (1987-1992) and MIT Press (1994).

Acknowledgement

I thank my colleagues at the Logic Systems Laboratory and at the University of Geneva for the stimulating atmosphere that made this work possible. Also thanks to the Editor of this series for his patience and advise and to my colleague A. Tettamanzi for critically reading the manuscript.

References

1. G. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Reading, MA, 1989.

2. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Second Edition, Berlin, 1994.

3. D.B. Fogel, *An Introduction to Simulated Evolutionary Optimization*, IEEE Trans. on Neural Networks, **5**, 3, 1994.

4. J.P. Cohoon, S.U. Hegde, W.N. Martin and D. Richards: *Punctuated Equilibria: a Parallel genetic Algorithm*, in Proc. of the Second Int. Conf. on Genetic Algorithms, J.J Grefenstette (Editor), Lawrence Erlbaum Associates, 148, 1987.

5. R. Tanese: *Parallel genetic Algorithm for a Hypercube*, in Proc. of the Second Int. Conf. on Genetic Algorithms, J.J Grefenstette (Editor), Lawrence Erlbaum Associates, 177, 1987.

6. B. Manderick and P. Spiessens: *Fine-Grained Parallel Genetic Algorithms*, in Proc. of the Third Int. Conf. on Genetic Algorithms, J. D. Schaffer (Editor), Morgan Kauffman, 428, 1989.

7. T. Starkweather, D. Whitley and K. Mathias: *Optimization Using Distributed Genetic Algorithms*, in Parallel Problem Solving from Nature, Lecture Notes in Computer Science Vol. 496, H.-P. Schwefel and R. Männer (Editors), Springer-Verlag, 176, 1991.

8. M. Tomassini, *The Parallel Genetic Cellular Automata: Application to Global Function Optimization*, Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, Springer-Verlag, Wien, 385, 1993.

9. H. Mühlenbein, M. Schomish and J. Born, *The Parallel Genetic Algorithm as Function Optimizer*, Parallel Comput. **17**, 619, 1991.

10. T. Bäck, F. Hoffmeister and H.P. Schwefel, *A Survey of Evolution Strategies*, Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1991.

11. D. B. Fogel and J.W. Atmar, *Comparing genetic Operators with Gaussian Mutations in Simulated Evolutionary Processes using Linear Systems*, Biol. Cybern. **63**, 111, 1990.

12 J. J. Grefenstette, *A Users's Guide to GENESIS Version 5.0*, 1990.

13. G. J. E. Rawlins (Editor), *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA, 1991.

14. L. D. Whitley (Editor), *Foundations of Genetic Algorithms 2*, Morgan Kaufmann Publishers, San Mateo, CA, 1993.

15 C. H. Papadimitriu and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

16. L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand, New York, 1991.

17. J. Koza, *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

18. C. Cotta, J.F. Aldana, A.J. Nebro and J.M. Troya, *Hybridizing Genetic Algorithms with Branch and Bound Techniques for the Resolution of the TSP*, in Proceedings of the Int. Conf. on Artificial Neural Nets and Genetic Algorithms, D.W. Pearson, N.C. Steele and R.F. Albrecht (Editors), Springer-Verlag, 277, 1995.

19. T. Kido, H. Kitano and M. Nakanishi, *A Hybrid Search for Genetic Algorithms: Combining Genetic Algorithms, Tabu Search and Simulated Annealing*, in Proceedings of the fifth Int. Conf. on Genetic Algorithms, S. Forrest (Editor), Morgan Kaufmann, CA, 641, 1993.

20. M.F. Bramlette and E..E. Bouchard, *Genetic Algorithms in Parametric Design of Aircraft*, in ref. 16, 109, 1991.

21. D. Powell and M.M. Skolnik, *Using Genetic Algorithms in Engineering Design Optimization with Non-Linear Constraints*, in Proceedings of the fifth Int. Conf. on Genetic Algorithms, S. Forrest (Editor), Morgan Kaufmann, CA, 424, 1993.

22. S.G. Roberts and M. Turega, *Evolving Neural Networks: an Evaluation of Encoding Techniques*, in Proceedings of the Int. Conf. on Artificial Neural Nets and Genetic Algorithms, D.W. Pearson, N.C. Steele and R.F. Albrecht (Editors), Springer-Verlag, 96, 1995.

23. O. Pictet, M. Dacorogna, B. Chopard, M. Oussaidene, R. Schirru and M. Tomassini, *Using Genetic Algorithms for Robust Optimization in Financial Applications*, to appear, Neural network World, 1995.

24. R.A. Brooks and P. Maes (Editors), *Artificial Life IV*, MIT Press, Cambridge, MA, 1994.

25. L. Davis, D. Orvosh, JL. Cox and Y. Qiu, *A Genetic Algorithm for Survivable Network Design*, in Proceedings of the fifth Int. Conf. on Genetic Algorithms, S. Forrest (Editor), Morgan Kaufmann, CA, 408, 1993.

26. J.T. Alander, *An Indexed Bibliography of Genetic Algorithms: Years 1957-1993*, Technical Report N.94-1, University of Vaasa, Finland, 1994. The bibliography can also be obtained by anonymous ftp as several compressed postscript files from: ftp.uwasa.fi in directory: cs/report94-1.

27. S. Wolfram *Cellular Automata and Complexity*, Addison-Wesley, Reading, MA, 1994.

28. R. Das, M. Mitchell and J.P. Crutchfield, *A Genetic Algorithm Discovers Particle-Based Computation in Cellular Automata*, in Parallel Problem Solving from Nature, Lecture Notes in Computer Science 866,

Y. Davidor, H.-P. Schwefel and R. M. Männer (Editors), Springer-Verlag, 344, 1994.

29. A. Loraschi, A. Tettamanzi, M. Tomassini and P. Verda, *Distributed Genetic Algorithms with an Application to Portfolio Selection Problems*, in Proceedings of the Int. Conf. on Artificial Neural Nets and Genetic Algorithms, D.W. Pearson, N.C. Steele and R.F. Albrecht (Editors), Springer-Verlag, 384, 1995.

30. K.E. Kinnear (Editor), *Advances in Genetic Programming*, MIT Press, Cambridge, MA, 1994.

31. D.B. Fogel, *Evolutionary Computation*, IEEE Press, New York, 1995.