White Paper

Title: Evolutionary Computation: a 2-page Overview for Practitioners

Status: DRAFT

Author: Giovanni Squillero
giovanni.squillero@polito.it

http://www.cad.polito.it/

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>3-Oct-11</td>
<td>Document created</td>
</tr>
</tbody>
</table>
Evolutionary Computation
A Two-Page Overview for Practitioners

Giovanni Squillero
giovanni.squillero@polito.it
http://www.cad.polito.it/

Abstract— Evolutionary heuristics have been little, but steadily, exploited during the past 20 years for solving difficult problems. This short introductions sketches the origin and principles of Evolutionary Computation, and outlines some (very) common pitfalls in its application.

I. EVOLUTIONARY COMPUTATION

Evolution is the biological theory that animals and plants have their origin in other types, and that the distinguishable differences are due to modifications in successive generations [1]. Natural evolution is not a random process. On the contrary, it is based on random variations, but some are rejected while others preserved according to objective evaluations. Only changes that are beneficial to the individuals are likely to spread into subsequent generations. Darwin called this principle “natural selection” [2], a quite simple process where random variations “afford materials”.

When natural selection causes variations to be accumulated in one specific direction the result strikingly resembles an optimization process. This process only requires to assess the effect of random changes, not the ability to design intelligent modifications. Several researchers, independently, tried to replicate such a characteristic to solve difficult problems more efficiently.

Evolutionary computation (EC) is the offshoot of computer science focusing on algorithms loosely inspired by the theory of evolution. The definition is deliberately vague since the boundaries of the field are not, and cannot be, sharply defined. EC is a branch of computational intelligence, and it is also included into the broad framework of bio-inspired heuristics.

EC does not have a single recognizable origin. Some scholars identify its starting point in 1950, when Alan Turing drew attention to the similarities between learning and evolution [3]. Others pointed out the inspiring ideas that appeared later in the decade, despite the fact that the lack of computational power impaired their diffusion in the broader scientific community [4]. More commonly, the birth of EC is set in the 1960s with the appearance of three independent research lines: John Holland’s genetic algorithms (GA) [5]; Lawrence Fogel’s evolutionary programming (EP) [6]; Ingo Rechenberg’s and Hans-Paul Schwefel’s evolution strategies (ES) [7] [8]. These three paradigms monopolized the field until the 1990s, when John Koza entered the arena with genetic programming (GP) [9]. These four, together with the many other different EC paradigms that have been proposed over the years, can be grouped under the term evolutionary algorithms (EAs).

In EAs a single candidate solution is termed individual; the set of all candidate solutions that exists at a particular time is called population. Evolution proceeds through discrete steps called generations. In each of them, the population is first expanded and then collapsed, mimicking the processes of breeding and struggling for survival. Some evolutionary algorithms do not store a collection of distinct individuals, and evolution is depicted through the variation of the statistical parameters that describe the population.

Most of the jargon of evolutionary computation mimics the precise terminology of biology. The ability of an individual to solve the target problem is measured by the fitness function, which influences the likelihood of a solution to propagate its characteristics to the next generation. In some approaches individuals may die of old age, while in other they remain in the population until replaced by fitter ones.

The word genome denotes the whole genetic material of the organism, although its actual implementation strongly differs from one approach to another. The gene is the functional unit of inheritance, or, operatively, the smallest fragment of the genome that may be modified in the evolution process. Genes are positioned in the genome at specific positions called loci. The alternative genes that may occur at a given locus are called alleles.

Biologists need to distinguish between the genotype and the phenotype: the former is all the genetic constitution of an organism; the latter is the set of observable properties that are produced by the interaction between the genotype and the environment. In many implementations, EC practitioners do not require such a precise distinction. The single numerical value representing the fitness of an individual is sometimes assimilated to its phenotype.

To generate the offspring, EAs implement sexual and asexual reproduction. The former is named recombination; it involves two or more participants, and implies the possibility for the offspring to inherit different characteristics from different parents. When recombination is achieved through an exchange of genetic material between the parents, it often takes the name of crossover. Asexual reproduction may be named replication, to indicate that a copy of an individual is created, or, more commonly, mutation, to stress that the copy is not exact. All operators exploited during reproduction can be cumulatively called evolutionary operators, or genetic operators because they act at the genotypical level. Almost no evolutionary algorithm takes gender into account; hence, individuals do not have distinct reproductive roles.

II. PRACTICAL USE OF EVOLUTIONARY COMPUTATION & COMMON PITFALLS

An EA performs better than a pure random approach. This rather simple consideration is probably the main reason why EAs are sometimes exploited outside the EC community. They provides an effective methodology for trying random modifications, where no preconceived idea about the optimal
solution is required. Being based on a population, EAs are more robust than pure hill climbing. Both small and large modifications are possible, but with different probabilities. Sexual recombination allows merging useful characteristics from different solutions, exploring efficiently the search space. Furthermore, EAs are quite simple to set up, and require no human intervention when running. They are inherently parallel, and a nearly-linear speed-up may be easily achieved on multiple instruction/multiple data (MIMD) architectures. Finally, it’s easy to trade-off between computational resources and quality of the results.

Unfortunately, several hidden and rather obscure details may significantly impair EAs’ efficacy. This may explain the relative slow acceptance, compared to other bio-inspired heuristics, such as simulated annealing (SA). First, failing to define an appropriate fitness function may easily spoil the evolution process. The fitness function must be able to discriminate between almost all individuals. That is, different solutions must receive different fitness values, because small variations must trigger differential survival in the artificial population.

Moreover, since the fitness is a synthetic measure, it is necessary to carefully consider which information is removed. For instance, if an EA is used to implement an automatic test-pattern generator (ATPG) the fitness cannot be simply defined as the attained fault coverage (FC%), because two individuals detecting 10% of the faults would be considered absolutely equivalent. However, if the first one is able to detect some random-resistant, hard-to-test faults while the second is an uninteresting random pattern, only the characteristics of the first one must be preserved in subsequent generations.

The choice of the representation for the individual is also important. Individuals must encode some structural information about the desired solution. However, too much information may bias the process preventing the discovery of optimal solutions. On the contrary, too few information may increase the dimension of the search space slowing down the evolution process. In some case, the encoding follows naturally from the problem. For instance, a fixed number of real parameters when optimizing filter coefficients, or a variable number of bit vectors when implementing a sequential ATPG. Conversely, different problems, like the design of circuits or the creation of assembly-language programs, require less obvious encoding. The individual representation also influences the behavior of genetic operators. Evolution calls for the offspring to preserve most of the parents’ characteristics, however they are defined by the current application.

Besides, the main problem with EA is definitely premature convergence. Generation after generation individuals in the population tend to be all alike, the population converges to a single point in the search space, all recombination operators become quite ineffective and the evolution process almost stops. In such a condition, an EA behaves like an overloaded, inefficient random-mutation hillclimber.

Premature convergence is an endemic problem of EC. In nature, “natural selection, also, leads to divergence of character: for more living beings can be supported on the same area the more they diverge in structure, habits, and constitution” [2]. However, in artificial evolution there is no explicit environment since its effects are modeled through the fitness function. Consequently, the divergence of character, i.e., one of the pillar of the Darwinian theory of evolution, is completely missing.

Several contributions describe methodologies to tackle this problem [10] [11] [12]. Authors proposed to preserve the diversity of the population by aging individuals, limiting interactions or other quite complex mechanisms. However, none of these artificial mechanisms can be called a panacea, and practitioners still need to tackle premature convergence on a problem-specific basis.

### III. REFERENCES


