Metaheuristics Set

coordinated by
Nicolas Monmarché and Patrick Siarry

Volume 9

Evolutionary Algorithms

Alain Pétrowski
Sana Ben-Hamida

iSTE

WILEY

First published 2017 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

Library of Congress Control Number: 2017931831

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-84821-804-8

Contents

Preface

Evolutionary algorithms are expected to provide non-optimal but good quality solutions to problems whose resolution is impracticable by exact methods. They are inspired by Darwin's theory of natural selection.

This book is intended for readers who wish to acquire the essential knowledge required to efficiently implement evolutionary algorithms. The vision offered by this book is essentially algorithmic and empirical. Indeed, the theoretical contributions in this field are still too incomplete to provide significant help in solving the problems for which these algorithms are used.

Chapter 1 describes a generic evolutionary algorithm as well as the basic operators that compose it. The main concepts presented in the other chapters are introduced here. This chapter also presents the binary representation and introduces *Genetic Algorithms*.

Chapter 2 is devoted to the solving of continuous optimization problems, without constraint. Real representation is introduced as well as basic variation operators able to deal with it. Three efficient approaches are then described: *Covariance matrix Adaptation Evolution Strategies*, *Differential Evolution* and *Particle Swarm Optimization*, which are well suited for continuous optimization. The chapter ends with comparisons between these approaches on a set of test functions.

Chapter 3 supplements Chapter 2 by adding constraints to an objective function defined in a continuous search space. The different approaches to taking constraints into account are presented with their main variants. Some methods are related to multi-objective optimization, which is presented in Chapter 5.

Chapter 4 is related to combinatorial optimization. These problems are extremely varied and often very difficult. From examples of resolution, it is difficult or even impossible to deduce information that is generalizable to other cases. Thus, we have decided to present a catalog of variation operators able to deal with the constraints associated with order-based problems. This class of problems has been chosen because they are often parts of real-world problems. For this reason, they have been the subject of an important research effort for several decades.

Chapter 5 first presents the basic notions necessary to understand the issue of multi-objective optimization. It then describes the NSGA-II method, which is one of the best known and most effective in the field. But NSGA-II, like the other algorithms based on Pareto dominance, is efficient when there are less than 4 objectives. So, the chapter ends with a presentation of a range of approaches to solve problems with many objectives, i.e. 4 or more.

Chapter 6 is devoted to the description of different approaches of genetic programming used in the context of machine learning. Two classes of applications are presented: symbolic regression and symbolic classification. Genetic programming is the subject of a special chapter to emphasize the ability of evolutionary algorithms to discover good solutions in an ill-defined search space, whose size is not fixed during an evolution.

Acknowledgments

We warmly thank Professors Patrick Siarry and Nicolas Monmarché for inviting us to contribute to the "Metaheuristics" set of books. Professor Michalewicz agreed to give us valuable advice when we developed the contents of this book. We are indebted to him for his help and we thank him with gratitude.

Alain PÉTROWSKI
Sana BEN-HAMIDA
February 2017

1

Evolutionary Algorithms

This chapter presents the basic principles of evolutionary algorithms as an introduction to the subsequent chapters. After a brief history of the domain in section 1.1, a generic evolutionary model is described in section 1.2. Sections 1.3 to 1.5 detail widespread variants of the operators composing the evolutionary algorithms, with a particular emphasis on binary representation. The chapter ends with a short presentation in section 1.6 of the famous *genetic algorithms* that have the originality to favor the binary representation associated with a transcription genotype–phenotype.

1.1. From natural evolution to engineering

According to Charles Darwin [DAR 59], the evolution of living beings rests on several facts:

– the variations of individual characteristics between parents and offspring;

– the heritability of much of these characteristics;

– a competition that selects the fittest individuals of a population in relation to their environment, in order to survive and reproduce.

From these facts, Darwin deduced that competition allows the transmission of hereditary beneficial variations among individuals that accumulate from generation to generation.

In the 1950s, the development of the electronic computer facilitated the simulation of this theory and some researchers desired to test it to solve engineering problems. But these works were not convincing because of the weak performances of the calculators available at that time. Furthermore, the extreme slowness of the evolution appeared prohibitive to usefully simulate this process.

During the 1960s and 1970s, as soon as calculators of more credible capacity became accessible, many attempts to model the process of evolution were undertaken. Among those, three approaches emerged and progressed independently until the beginning of the 1990s:

– the *evolution strategies (ESs)* of H. P. Schwefel and I. Rechenberg [REC 65, BEY 01], which are derived from an experimental optimization method to solve fluid dynamics problems;

– the *evolutionary programming* (EP) of L. J. Fogel *et al.* [FOG 66] which aimed, in the mid-1960s, to evolve the structure of finite-state automata with iterated selections and mutations; it was desired to be an alternative to artificial intelligence at the time;

– *Genetic algorithms* (GAs) were presented in 1975 by J.H. Holland [HOL 92], with the objective of understanding the underlying mechanisms of systems able to self-adapt to their environment.

Thereafter, these approaches underwent many modifications according to the variety of the problems faced by their founders and their pupils. Genetic algorithms became extremely popular after the publication of the book "*Genetic Algorithms in Search, Optimization and Machine Learning*" by D. E. Goldberg in 1989 [GOL 89]. This book, distributed worldwide, resulted in exponential growth in interest in this field. While there were about a few hundred publications in this area during the 20 year period before this book was published, there are several hundreds of thousands of references related to evolutionary computation available today, according to the website "google scholar"[1]. Researchers in this field have organized common international conferences for presenting and combining their different approaches.

The widespread term *Evolutionary Computation* appeared in 1993 as the title of a new journal published by the MIT Press. It was widely used to designate all methods based on the metaphor of the biological evolution

1 https://scholar.google.com/scholar?q="genetic+algorithms"

theory, as well as many others. For example, although it is inspired by a simplified model of social behavior, it is common to consider “Particle Swarm Optimization” as an evolutionary approach. “Particle Swarm Optimization” algorithms have notable common points with other more conventional evolutionary algorithms by assimilating “swarm” and “population”, “particles” and “individuals”. According to Michalewicz [MIC 12]: “*It seems that Evolutionary Algorithms, in the broad sense of this term, provide just a general framework on how to approach complex problems.*”

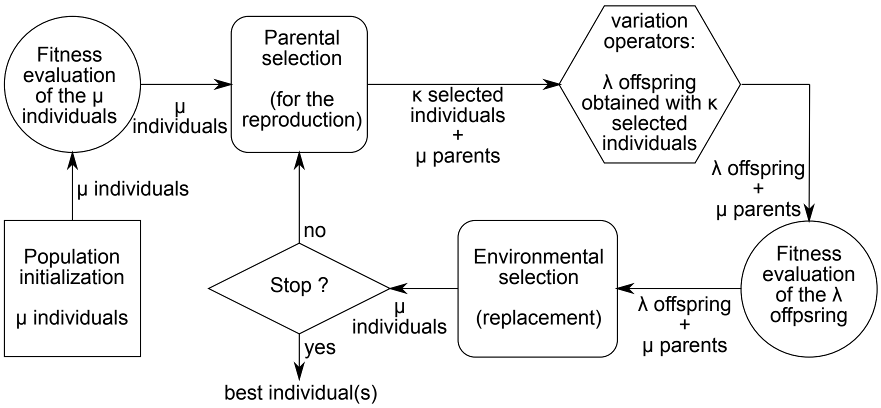
1.2. A generic evolutionary algorithm

Evolutionary algorithms are iterative algorithms that make the individuals of a population evolve over a number of *generations*. A generation is in fact an iteration of the main loop of these algorithms (Figure 1.1). Any individual in a population contains at least the information required to represent a more or less efficient solution to the target problem or a part of a solution, such as in “Learning Classifier Systems” [BOO 89] [URB 09].

An evolutionary algorithm works on a population of interacting *parent* individuals to produce a set of *offspring* individuals at every *generation*. At the end of each generation, selected offspring replace parents to build the parent population at the next generation, in such a way that its size is constant or, more rarely, controlled according to a given policy.

In accordance with the Darwinist guiding principle, the fitter an individual is, the more often it is selected to reproduce or survive. To make this selection possible, a *fitness* value is attached to each individual. It may be evaluated with a given *fitness function*, or possibly by other means such as simulations. For instance, in the context of optimization tasks, the fitness function is the objective function or else it strongly depends on the objective function. For each generation, the fitnesses of the offspring have to be evaluated, which can be computation intensive if the population is large.

The *variation operators* generate new offspring from one or several parents according to a given *representation*. For instance, to solve a given problem in a domain of $\mathbb{R}^n$, this representation could be an array of n floating point numbers. But it could also be an array of binary digits that represents a vector of these n real values, with an appropriate encoding. Thus, the choice of a representation for the individuals depends not only on the problem, but also on the properties of the available variation operators. These properties have to favor the generation of good offspring as much as possible. The choice of appropriate representation and variation operators has a critical influence on the performance of evolutionary algorithms.



Fitness evaluation of the μ individuals
μ individuals
Parental selection (for the reproduction)
κ selected individuals + μ parents
variation operators: λ offspring obtained with κ selected individuals
λ offspring + μ parents
Fitness evaluation of the λ offpsring
λ offspring + μ parents
Environmental selection (replacement)
μ individuals
Stop ?
no
yes
best individual(s)
μ individuals
Population initialization μ individuals

Figure 1.1. *The generic evolutionary algorithm*

Interactions between individuals of a population characterize the evolutionary algorithms. Several parents can interact to generate new offspring. Offspring and parents interact to select the fittest of them for reproduction or survival through *election operators*. Obviously, when a population contains only one parent, for instance, there is no interaction between several parents. However, this particular case is still an evolutionary algorithm if the interactions are specified in the algorithm and occur when they can be applied.

Figure 1.1 depicts the generic evolutionary algorithm. The operators involved in this algorithm are generic because many variants of these operators have been proposed:

– first, the initialization step generates μ individuals to constitute a population, μ being a free parameter of the algorithm. The population initialization can be obtained from dedicated heuristics if some information about the problem to solve is known, or in the contrary case, by default, the initialization step generates the individuals randomly in the search domain according to a distribution depending on the evolutionary algorithm variant used;

– the fitness values of the μ individuals are then determined during the "fitness evaluation" step;

– the *parental selection* or *selection for the reproduction*, or simply the *selection* operator determines how many times any individual will be reproduced in a generation, depending on its fitness: the higher the fitness of an individual, the more it reproduces. The selection operator generates a total of κ copies of parents that will be used by the variation operators;

– assuming that each selected copy can be used only once by the variation operators, κ has to be large enough to allow the variation operators to generate λ offspring, where λ is another free parameter of the algorithm. For instance, if the variation operators produce two offspring from one pair of parents, then $\kappa = \lambda$. If the variation operators produce only one offspring from one pair of parents, then λ pairs of parents are required and $\kappa = 2\lambda$. The hexagonal shape related to "variation operators" shown in Figure 1.1 may represent a chain of several operators that are subsequently applied, such as mutation and crossover operators, typically;

– the fitnesses of the λ offspring are then evaluated;

– afterwards, the *environmental selection* or *selection for the replacement*, or simply the *replacement* operator decides which individuals will survive in the population in the next generation among the μ parents and λ offspring;

– when the current generation ends, a stopping test allows the algorithm to stop according to the criteria defined by the user. Otherwise, another generation begins with the new population built by the environmental selection operator.

Among the evolutionary algorithm variants, parental selection or else environmental selection may be independent of fitnesses, provided that there is a bias in favor of the fittest individuals from generation to generation.

1.3. Selection operators

1.3.1. *Selection pressure*

The individuals which have the best fitnesses are reproduced more often than the others and replace the worst ones. Sometimes, a population contains a non-optimal super-individual with a much higher fitness than the others.

Depending on the nature or the parameters of the selection operators, it could potentially reproduce much more quickly than the others. Its copies could then invade the population before the variation operators find better solutions. The exploration of the search space becomes local, since it is limited to a random local search centered on the super-individual. In this way, there will be a high risk that the algorithm cannot find a global optimum because it remains trapped in any local optimum.

In another case, according to the nature of the parameters of the selection operators, low fitness individuals and high fitness individuals could have almost the same average number of offspring. In this case, the convergence could be strongly slowed down unnecessarily.

In the first case mentioned above, the super-individual creates a selection pressure that is too high, while in the second case, the selection pressure is too low.

If the variation operators are disabled, the best individual should reproduce more quickly than the others, until its copies completely take over the population. This observation leads to the first evaluation of the selection pressure for evolutionary algorithms, suggested by Goldberg and Deb in 1991 [GOL 91]. The *takeover time* τ^* is defined as the number of generations required to fill the population with copies of the best individual under the action of the selection operators only: the lower the value of τ^*, the higher the selection pressure.

The *selection intensity* S is another concept, borrowed from the "population genetics" theory [HAR 06], to evaluate the selection pressure. Let $\bar{f}$ be the average fitness of the population before the selection. Let $\bar{g}$ be the average fitness of the selected individuals. Then, S measures the increase in the average fitness of the individuals of a population determined before and after selection with the standard deviation σ_f of the individual fitnesses before selection taken as a unit of measure:

$$S = \frac{\bar{g} - \bar{f}}{\sigma_f}$$

If the selection intensity is computed for the reproduction, then $\bar{f} = \sum_{i=1}^{\mu} f_i/\mu$, where f_i is the fitness of an individual i, and $\bar{g} = \sum_{i=1}^{\kappa} g_i/\kappa$, where g_i is the fitness of the selected individual i and κ is the number of selected individuals. The definitions presented above are general and are applicable to any selection technique. It is possible to present other definitions, whose validity is possibly limited to certain techniques, as we will see later with regard to the *proportional selection*.

1.3.2. *Genetic drift*

Genetic drift is also a concept which derives from the *population genetics* theory [HAR 06]. It is related to random fluctuations of the frequency of alleles in a population of small size, where an *allele* is a variant of an element of a sequence of DNA having a specific function. For this reason, hereditary features can disappear or be fixed at random in a small population even without any selection pressure.

This phenomenon also occurs within the framework of the evolutionary algorithms if their selection operators are stochastic. At the limit, even for a population composed of different individuals subject to neutral selection[2], in the absence of the variations generated by mutation and crossover operators, the population converges towards a state where all the individuals are identical. Figure 1.2 illustrates the effect of genetic drift in a population of 100 individuals in a two-dimensional search space Ω, according to the generation number. Individuals are represented as points, possibly "stacked" as vertical lines when several individuals are at the same location in Ω. Individuals are subject to neutral selection, without mutation or crossover. The figure shows that at the 100th generation, all the individuals are identical.

The genetic drift can be estimated from the time required to obtain a homogeneous population using a Markovian analysis. But these results are approximations and are difficult to generalize outside the cases studied in the literature [ROG 99]. However, it is verified that the time of convergence towards an absorbing state becomes longer as the population size increases.

Another approach assesses the genetic drift rate caused by a selection operator [ROG 99]. Let r be the ratio of $E(V'_f)$: the expectation of the fitness variance of the population after selection, to V_f: the fitness variance before

2 Neutral selection: the number of offspring of an individual is independent of its fitness.

selection. In the case of neutral selection, A. Rogers and A. Prügel-Bennett showed that r depends only on variance V_s of the number of offspring for each individual and on population size P, assumed constant:

$$r = \frac{E(V_f')}{V_f} = 1 - \frac{V_s}{P-1}$$

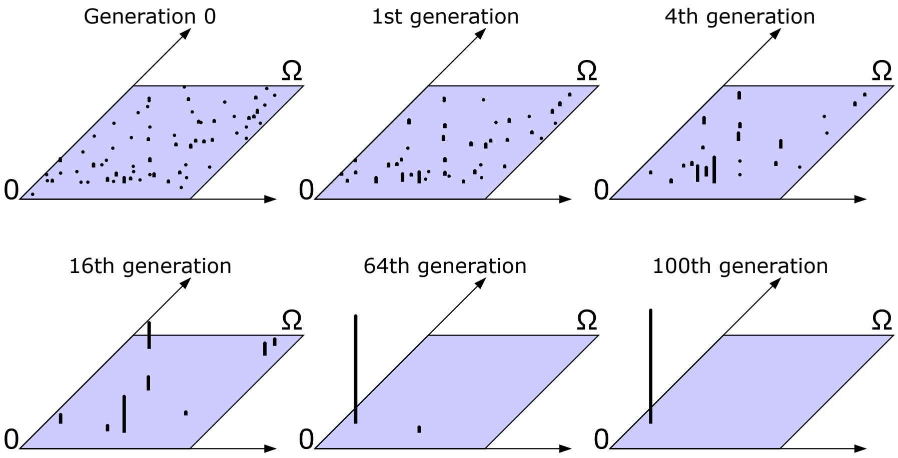


Figure 1.2. *Effect of genetic drift according to the generation number, for a population of 100 individuals in a two-dimensional search space* Ω

V_s is a characteristic of the selection operator. The expression clearly shows that increasing the population size or reducing the variance V_s of the selection operator decreases r, i.e. the genetic drift. Figure 1.3 shows the effect of genetic drift with the same conditions as mentioned previously (Figure 1.2), but for a population of 1,000 individuals. After 100 generations, there is still diversity in the population of 1,000 individuals, whereas the population of size 100 only contains identical individuals.

The effect of genetic drift is predominant when the selection pressure is low. This situation leads to a loss of diversity, which may involve a premature convergence *a priori* far away from the optimum, since it does not depend on the fitness of the individuals.

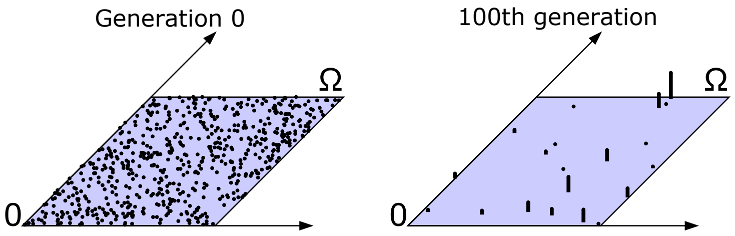


Figure 1.3. *Effect of genetic drift according to the generation number for a population of 1,000 individuals in a two-dimensional search space* Ω

In short, in order for an evolutionary algorithm to work efficiently, it is necessary that the selection pressure is neither too strong nor too weak. To fight the genetic drift, the population size must be large enough, unless the selection operator is characterized by a low variance.

1.3.3. *Proportional selection*

Proportional selection was originally proposed by J. Holland for genetic algorithms. It is used only for the reproduction. The expected number of selections n_i of an individual i is proportional to its fitness f_i. This implies that the fitness function is positive in the search domain and that it has to be maximized. Let μ be the population size and let κ be the total number of individuals generated by the selection operator for one generation, n_i can be expressed as:

$$n_i = \frac{\kappa}{\sum_{j=1}^{\mu} f_j} f_i$$

However, the effective number of offspring can be only integers. They can be obtained by successive random draws of offspring according to distributions chosen such that the expected number of offspring for parent i is equal to n_i. Two techniques are widespread and are described below: the *roulette wheel selection method, RWS* because it is the operator originally proposed for the genetic algorithms, but it suffers from high variance, and the *stochastic universal sampling method, SUS* because it guarantees a minimal variance of the sampling process [BAK 87].

1.3.3.1. *Roulette Wheel Selection (RWS)*

The RWS method exploits the metaphor of a biased roulette game, for which the wheel comprises as many pockets as individuals in the population. Each individual is associated to a pocket with a size proportional to its fitness. Once the game is started, the selection of an individual is indicated by the fall of the ball into its pocket. For instance, Figure 1.4 depicts a roulette wheel where the ball has fallen into pocket 7. So, individual 7 with fitness 156 is selected for the reproduction. The ball is spun κ times to select κ individuals.

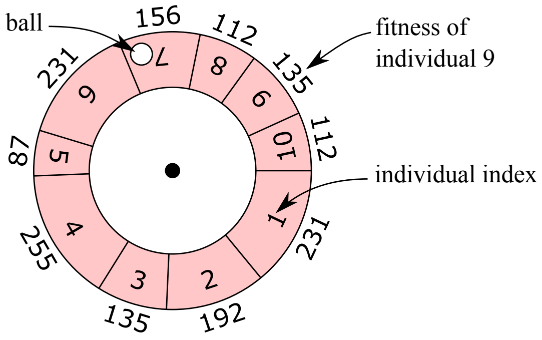


Figure 1.4. *Biased roulette wheel metaphor for the RWS method: individual 7 with fitness 156 is selected after drawing a random number represented as the ball position on a roulette wheel*

The RWS algorithm selects individual i such that:

$$\sum_{j=0}^{i-1} f_j \leq \mathcal{U}[0,1] \sum_{j=1}^{\mu} f_j < \sum_{j=1}^{i} f_j \quad \text{with} \quad f_0 = 0$$

where $\mathcal{U}[0,1]$ is a random number uniformly drawn from [0, 1]. f_j is the fitness of individual j.

1.3.3.1.1. Genetic drift

The effective number of selections of individual i follows a binomial distribution $\mathcal{B}(\kappa, p_i)$ with $p_i = f_i / \sum_{j=1}^{\mu} f_j$. Its variance $V_{\text{RWS}}(i)$ is $\kappa p_i(1-p_i) = n_i(1-p_i)$, where n_i is the expected number of individual i selections. For a given selection pressure (section 1.3.3.3), and if μ is large enough, $V_{\text{RWS}}(i) \approx n_i$. The variance of this process is high: it is possible that an individual with a good fitness value is never selected. By misfortune, it is also possible that bad quality individuals are selected for all the offspring. Thus, the high variance creates a high genetic drift level, allowing some poor individuals to have offspring to the detriment of better individuals. To reduce this risk, the population size must be large enough.

1.3.3.1.2. Algorithmic complexity

The selection of an individual requires $\mathcal{O}\log(n)$ comparisons by using a dichotomy algorithm, where $n = \mu + \kappa$. In this way, there are $\mathcal{O}(n\log(n))$ comparisons to select κ individuals.

1.3.3.2. *Stochastic Universal Sampling selection (SUS)*

The SUS method [BAK 87] still uses the metaphor of the wheel of a biased roulette game. However, the ball is spun only once to obtain the set of the κ selected individuals. When the ball falls in a pocket, it defines a random offset position. Offspring are then determined by a set of κ equidistant pointers around the wheel such that one of these pointers is set at the offset position, as shown in Figure 1.5. According to the figure, individuals 5 and 9 are not selected, and the others are selected once.

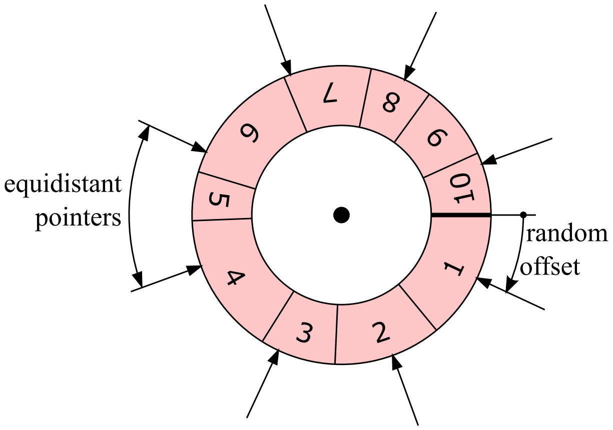


Figure 1.5. *SUS method: the selected individuals are determined by $\kappa = 8$ equidistant pointers. Thus, individuals 5 and 9 are not selected, and the others are selected once. The fitnesses corresponding to the individuals are given in Figure 1.4*

Let δ be the distance between the pointers: $\delta = \sum_{j=1}^{\mu} f_i/\kappa$. Let ω be the random offset: $\omega = \mathcal{U}[0, \delta]$. The SUS operator selects κ individuals i such that:

$$\forall k \in \{0, ..., \kappa - 1\}, \quad \sum_{j=0}^{i-1} f_j \leq \omega + k\delta < \sum_{j=1}^{i} f_j \quad \text{with} \quad f_0 = 0$$

1.3.3.2.1. Genetic drift

For an expected number of selections n_i of individual i, the effective number of selections is either its lower integer part $\lfloor n_i \rfloor$, or its upper integer part $\lceil n_i \rceil$ according to a Bernoulli distribution of parameter $q_i = n_i - \lfloor n_i \rfloor$, with variance $V_{\text{SUS}}(i) = q_i(1 - q_i)$. The maximum of V_{SUS} is 1/4 for $q_i = 1/2$. From expressions of $V_{\text{RWS}}(i)$ and $V_{\text{SUS}}(i)$, it can easily be shown that $V_{\text{SUS}}(i) < V_{\text{RWS}}(i)$ if $\kappa > 1$.

If $\kappa \geq \mu$, the best individual is certain to have at least one offspring. This is the case in Figure 1.6 where $\mu = \kappa = \lambda = 100$. When the variation operators are disabled, the population is kept unchanged indefinitely under SUS selection for a constant fitness function (neutral selection). In contrast, Figure 1.2 (p. 8) shows what happens during an evolution by replacing SUS selection with the RWS operator: its high variance results in a complete loss of diversity in the population after 100 generations.

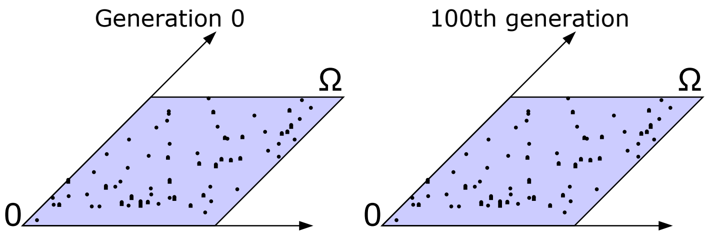


Figure 1.6. *Genetic drift does not occur with the SUS selection operator for a population of $\mu = \kappa = 100$ individuals in a two-dimensional search space Ω*

1.3.3.2.2. Algorithmic complexity

The number of comparisons to select the individuals is of the order of $\mathcal{O}(n)$, where $n = \mu + \kappa$.

1.3.3.3. *Proportional selection and selection pressure*

In the case of proportional selection, the expected number of selections of the best individual with fitness $\hat{f}$ among μ selections for a population of μ parents is appropriate to define selection pressure p_s:

$$p_s = \frac{\mu}{\sum_{j=1}^{\mu} f_j}\hat{f} = \frac{\hat{f}}{\bar{f}}$$

where $\bar{f}$ is the average of the fitnesses of the population. If $p_s = 1$, then all the individuals have equal chances to be selected, indicating an absence of selection pressure.

Let us consider the search for the maximum of a continuous function, e.g. $f(x) = \exp(-x^2)$. The individuals of the initial population are assumed to be uniformly distributed in the domain $[-2, +2]$. Some of them have a value close to 0, which is also the position of the optimum, and thus their fitness $\hat{f}$ will be close to 1. The average fitness of the population $\bar{f}$ will be:

$$\bar{f} \approx \int_{-\infty}^{+\infty} f(x)p(x)dx$$

where $p(x)$ is the presence probability density of an individual at x. A uniform density is chosen in the interval $[-2, +2]$, thus $p(x) = 1/4$ in this interval, and is 0 elsewhere. Thus,

$$\bar{f} \approx \frac{1}{4}\int_{-2}^{+2} e^{-x^2}dx$$

that is $\bar{f} \approx 0.441$, which gives a selection pressure of the order of $p_s = \hat{f}/\bar{f} \approx 2.27$. The best individual will thus have an expected number of offspring close to two at generation 0.

Figure 1.7 shows the repartition of a population of $\mu = \lambda = \kappa = 100$ individuals according to the number of generations to find the maximum of $f(x) = \exp(-x^2)$ in interval $[-2, 2]$. Individuals are represented as dots on the curve of $f(x)$. The figure also indicates the value of the fitness average for each graph by using a dashed line and the number of offspring n_{best} for the best individual taken as a measure of the selection pressure. Individuals are subject to a proportional selection (RWS) and uniform mutations in interval [–0.05, 0.05], i.e. $x' = x + \mathcal{U}[-0.05, 0.05]$, where x' is the offspring obtained from the mutation of x and $\mathcal{U}[-0.05, 0.05]$ is a uniform random number in $[-0.05, 0.05]$. The environmental selection replaces all the individuals in the population at the next generation with their offspring (generational replacement).

Figure 1.7 shows that when individuals concentrate towards the optimum over generations due to the selection pressure, the fitness average increases, approaching the maximum value and consequently, the selection pressure decreases, tending to one. This means that:

– every individual has an expected number of offspring close to one, even the best one;

– the stochastic fluctuations of the selection operator become predominant to determine the offspring number (genetic drift);

– the evolution is no longer guided by the fitness and therefore the algorithm fails to find the optimum with a high precision.

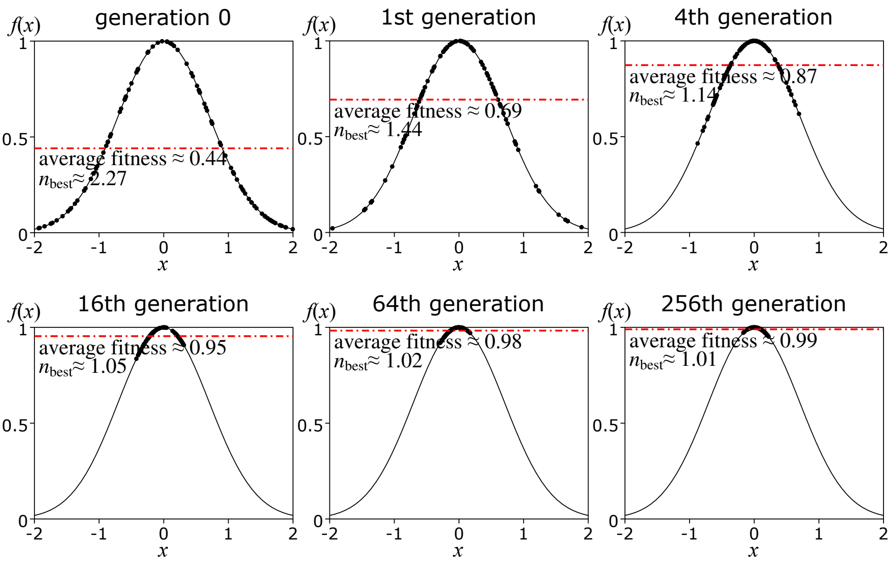


Figure 1.7. *Selection pressure n_{best} decreases when the population concentrates in the neighborhood of the optimum*

This undesirable behavior of proportional selection, where the selection pressure strongly decreases when the population approaches the optimum in the case of continuous functions, can be overcome by scaling the fitness function [GOL 89]. It is interesting to note the Boltzmann selection (De La Maza and Tidor 1993 [DE 93b]), because it makes a link with simulated annealing. The method uses a scaled fitness expressed as:

$$f_i' = \exp(f_i/T)$$

The value of parameter T, known as the “temperature”, determines the selection pressure. T is usually a decreasing function of the number of generations. Thus, the selection pressure increases with time.

1.3.3.4. *Rank-based selection*

Another way to control the selection pressure combines a proportional selection and a ranking of the individuals of the population. Individuals x are ranked from the best one (first) to the worst one (last), according to raw fitnesses $f(x)$. The actual fitness value f_r of each individual depends only on its rank r by decreasing value (see Figure 1.8) according to, for instance, the expression given below, which is usual:

$$f_r = \left(1 - \frac{r}{\mu}\right)^p$$

where μ is the number of parents, r is the rank of the individual considered in the population of the parents after ranking and p is an exponent which depends on the desired selection pressure. After ranking, a proportional selection is applied according to f_r. With our definition of the selection pressure $p_s = n_{\text{best}} = 1 + p$. Thus, p must be greater than 0 to obtain a selection pressure greater than 1. This ranking-based selection is interesting because:

– it is not affected by a constraint of sign: $f(x)$ can either be positive or negative;

– it is appropriate for a maximization problem as well as for a minimization problem, without any extra transformation;

– it does not consider the importance of the differences between the fitnesses of the individuals. Thus, a ranking-based selection method does not require exact knowledge of the objective function. It only needs to rank the individuals by comparing each one with the others.

These good properties mean that ranking-based selections are often preferred by the users of evolutionary algorithms when compared to fitness scaling methods. Linear ranking, for which $p = 1$, is quite a good choice by default.

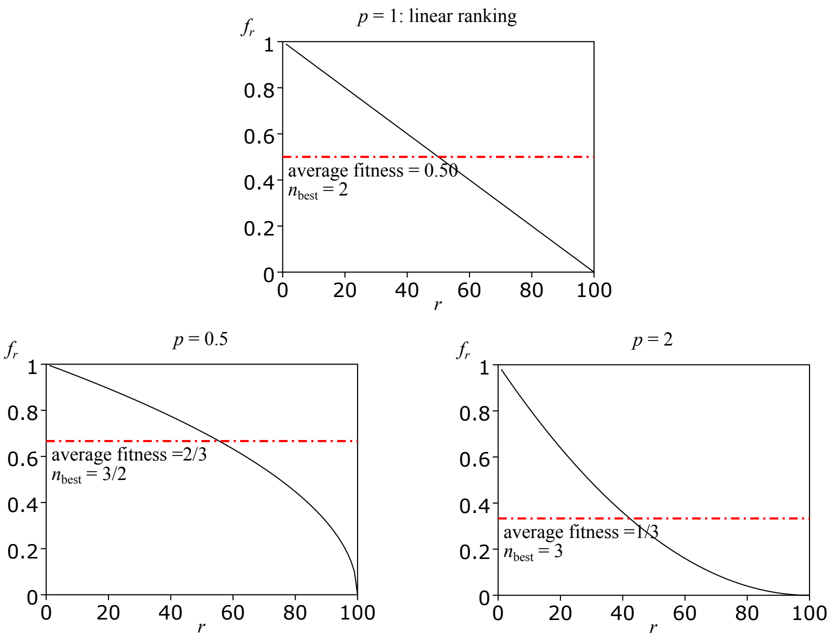


Figure 1.8. *Fitness $f_r = (1 - r/\mu)^p$ obtained after ranking, where r is the rank of an individual. μ is chosen equal to 100. The selection pressure is $n_{\text{best}} = 1 + p$*

1.3.4. *Tournament selection*

Tournament selection is an alternative to the proportional selection techniques which, as seen before, present difficulties in controlling the selection pressure during the evolution, while being relatively expensive in computational costs.

1.3.4.1. *Deterministic tournament*

The simplest tournament consists of choosing at random p individuals in the population and selecting the one that has the best fitness for reproduction.

During a selection step, there are as many tournaments as selected individuals. The individuals that take part in a tournament are replaced in the population, or withdrawn from it, according to the choice of the user. A drawing without replacement makes it possible to carry out $\lfloor N/p \rfloor$ tournaments for a population of N individuals. A copy of the population is re-generated when it is exhausted, as many times as required, until the desired number of selections is reached. The variance of the tournament process is high, which favors genetic drift. It is however lower in the case of drawing without replacement. This method of selection is very much used, because it is much simpler to implement than a proportional reproduction with properties similar to the ranking selection.

The selection pressure is adjusted by the number of contestants p in a tournament. Indeed, let us consider the case where the contestants in a tournament are replaced in the population. Then, the probability that the best individual of the population is not selected with p random drawings is $((N-1)/N)^p$. Assuming that N is large compared to p, this probability is approximately $1 - p/N$ by a Taylor expansion limited to the first order. Thus, the probability that the best individual is drawn at least once in a tournament is close to p/N. If there are κ tournaments in a generation, the best individual will have $n_{best} = p\kappa/N$ expected selections. Let us consider again the definition of selection pressure that was proposed for the proportional reproduction, with $\kappa = N$. Then, the selection pressure is equal to p, which is greater than or equal to 2.

1.3.4.2. *Stochastic binary tournament*

With the stochastic binary tournament, the best of the two contestants wins with a probability p, which is a parameter whose value is chosen between 0.5 and 1. It is still easy to calculate the selection pressure generated by this process. The best individual takes part in a tournament with a probability of $2/N$ (see the previous section 1.3.4.1). Furthermore, the winner of the tournament will be selected with a probability p. The two events being independent, the probability that the best individual of the population is selected after a tournament is then $2p/N$. If there are N tournaments, the best parent will thus have $2p$ expected offspring. The selection pressure thus will range between 1 and 2.

1.3.5. *Truncation selection*

This selection is very simple to implement, since it only chooses the n best individuals from a population, where n is a parameter chosen by the user. If the operator is used for the reproduction, $n = \kappa$ to select κ parents. If the operator is used for the replacement and thus generates the population of μ individuals for the next generation, then $n = \mu$.

1.3.6. *Environmental selection*

The *environmental selection* or *replacement selection* method determines which individuals in generation g, among offspring and parents, will constitute the population in generation $g + 1$.

1.3.6.1. *Generational replacement*

This kind of replacement is the simplest, since the population of the parents in generation $g + 1$ will be composed of all the offspring, and only them, generated at generation g, thus: $\mu = \lambda$ to keep the population size constant. The canonical genetic algorithm is proposed with a generational replacement.

1.3.6.2. *Replacement "(μ, λ) – ES"*

A truncation selection of the best μ individuals among the λ offspring constitutes the population for the next generation. This operator was introduced for the evolution strategies [REC 65, BEY 01]. In this case, λ is larger than μ.

1.3.6.3. *Replacement "$(\mu + \lambda)$ – ES"*

A truncation selection of the best μ individuals from the union of the set of the μ parents and the set of the λ offspring constitutes the population for the next generation. This operator was introduced for the evolution strategies [REC 65, BEY 01]. It is said to be elitist since it preserves the best individual found so far in the population.

1.3.6.4. *One-to-one selection*

This operator is a deterministic binary tournament typically between each parent of a population and its only offspring. The best of them is kept in the population for the next generation. Thus, the variance of the selection number for each parent is 0. In this context, the parental selection is disabled because each parent has only one offspring, whatever its fitness is. This very simple operator is especially used for the powerful *Differential Evolution* and *Particle Swarm Optimization* algorithms, described in Chapter 2.

1.3.6.5. *Steady state replacement*

In each generation, a few (often one or two) offspring are generated. They replace a lower or equal number of parents to produce the population at the next generation. This strategy is particularly useful when the representation of a solution is distributed on several individuals, possibly the entire population, as for *Learning Classifier Systems* [BOO 89] [URB 09]. In this way, the loss of a small number of individuals in each generation: those that are replaced by the offspring, does not excessively disturb the solutions, which evolve gradually.

The replaced parents can be chosen in various ways. With uniform replacement, the replaced parents are selected at random. The choice can also depend on the fitness: the worst parent is replaced, or else it is selected stochastically, according to a probability distribution that depends on the fitness or other criteria.

Steady state replacement generates a population where the individuals are subject to large variations of lifespan in terms of generation numbers and therefore offspring numbers. The high variance of these values involves high genetic drift, especially as the population is small [DE 93a].

1.3.6.6. *Elitism*

An elitist strategy consists of at least keeping the individual with the best fitness in the population, from generation to generation. The fitness of the best individual from the current population is thus monotonically non-decreasing from one generation to the next.

There are various elitist strategies such as the "$(\mu + \lambda)$ – ES" mentioned above. In another current alternative, the k best parents in the current generation are kept in the population for the next generation. The replacement operator then has to replace the $N - k$ remaining individuals, where N is the population size.

These strategies may considerably speed up the performance of evolutionary algorithms for some classes of fitness functions, such as convex functions. But, they may be disappointing for other classes, such as multimodal functions, because elitism increases the selection pressure around the best individuals of the population, even when they are trapped in local optima. However, this is an algorithm design issue. Thus, Chapter 2 presents elitist elaborated algorithms able to quite efficiently find near-optimal solutions for massively multimodal functions. Figure 1.9 shows a graphical representation in a 2D search space of such a multimodal function, which counts 15^d local minima in domain $[-40, 40]^d$, where dimension d is an integer usually taken between 2 and 100. Its analytic expression is given in section 2.10 (p. 89).

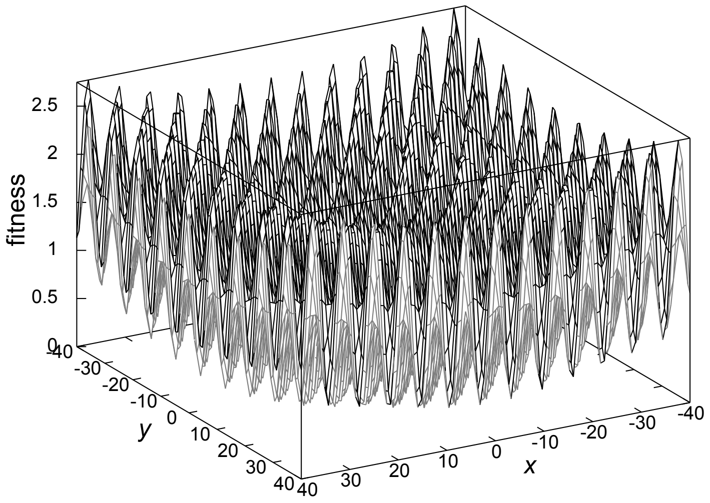


Figure 1.9. *The Griewank's multimodal function in domain* $[-40, 40]^2$

Choosing a non-elitist strategy can be advantageous, but there is no guarantee that the fitness function of the best individual in the current population is increasing during the evolution. This obviously implies keeping a copy of the best solution found so far by the algorithm, without this copy taking part in the evolutionary process. It should be noted, this is a requisite precaution for any stochastic optimization algorithm.

1.3.7. *Selection operators: conclusion*

Table 1.1 summarizes the properties of basic selection operators used by evolutionary algorithms in terms of selection pressure, variance of the selected individual number and algorithmic complexity. Among these operators, the best one is "one-to-one selection", but it is rather used by specific evolutionary algorithms (Chapter 2). The tournament selection is widespread because of its ease of implementation, despite a high variance. The population ranking associated with a selection process such as truncation selection or SUS is also often used, although its algorithm complexity is the worst.

	Selection pressure	Variance	Complexity
RWS	Non-controlled	High	$\mathcal{O}(n \log n)$
SUS	Non-controlled	Low	$\mathcal{O}(n)$
Tournament selection	Controlled	High	$\mathcal{O}(n)$
Ranking + SUS	Controlled	Low	$\mathcal{O}(n \log n)$
One-to-one selection	Controlled	0	$\mathcal{O}(n)$

Table 1.1. *Comparisons of basic selection operators according to their control of the selection pressure, variances and algorithmic complexities. n is the population size*

1.4. Variation operators and representation

1.4.1. *Generalities about the variation operators*

The variation operators are generally stochastic operators that transform and combine copies of one or several individuals in a population to create offspring, possibly fitter than their parents, which partly inherit the features of them. These operators are classified into two categories:

– the *mutation* operators, which alter an individual independently of the others;

– the *crossover* or *recombination* operators, which generate one or more offspring from the combinations of several parents, often two of them but also possibly the combination of the whole parent population.

The way of modifying an individual depends strongly on the structure of the solution that it represents. Thus, if we aim to solve an optimization problem in a continuous space, e.g. a domain of $\mathbb{R}^n$, *a priori*, it will be appropriate to choose a vector of $\mathbb{R}^n$ to represent a solution. In this case, the crossover operator combines at least two "parent" vectors of $\mathbb{R}^n$ to create one or several "offspring" vectors in the same space. On the other hand, if an evolutionary algorithm is used to solve instances of the *Traveling Salesman Problem*, an individual could represent a Hamiltonian path as a list of vertices. The variation operators should then generate only new Hamiltonian paths. These examples show that it is impossible to design universal variation operators, independently of the problem under consideration. They are inevitably related to the *representation* of the solutions in the search space.

Variation operators have to:

– *explore* the search space, in order to discover its promising areas, which are more likely to contain the global optima;

– *exploit* these promising areas, by focusing the search there to find the optima with the required accuracy.

For instance, a variation operator that draws offspring at random uniformly in the search space will have excellent qualities of exploration. But it will likely fail to discover an optimum in a reasonable time with the required accuracy, because there is no exploitation of the promising areas.

The search should be more efficient if the offspring are generated preferentially close to their parents, according to an appropriate distance, depending on the problem to solve. Such a policy is reasonable to the extent that parents are located in promising areas since they are selected according to their fitnesses. But if all the offspring generated by a variation operator are close to their parents, the search algorithm could be trapped in a local optimum: in this case, there is not enough exploration compared to exploitation.

A good balance should be found between exploitation and exploration abilities of variation operators. This requirement is not easy to ensure.

1.4.2. *Crossover*

The crossover operator uses at least two parents to generate one or more offspring. The *crossover rate*, which is *a priori* a parameter of the evolutionary algorithm, determines the proportion of the crossed individuals in the population. The operator is generally stochastic, and thus the repeated crossover with the same couple of distinct parents yields different offspring. It often respects the following properties:

– the crossover of two identical parents will generate offspring, identical to the parents;

– by extension, on the basis of an index of proximity depending on the chosen representation, defined in the search space, two parents which are close in the search space will generate offspring close to them.

These properties are satisfied by the "classical" crossover operators like most of those described in this book, but they are not absolute. In the current state of knowledge of evolutionary algorithms, the design of a crossover operator does not follow precise rules.

In the simplest version of an evolutionary algorithm, the selected individuals are mated at random. As a result, some selected parents could be located on several peaks of the fitness function, as depicted in Figure 1.10. The figure shows that the probability distribution of offspring is uniform between parent solutions x_1 and x_2. This widespread crossover is referred to as the flat crossover [RAD 90]. In this case, the conditional probability distribution of offspring given by the parents is inappropriate because offspring are likely to have poor fitness values. The crossover is said to be *lethal* if, from high fitness parents, it generates offspring with too low fitness to survive.

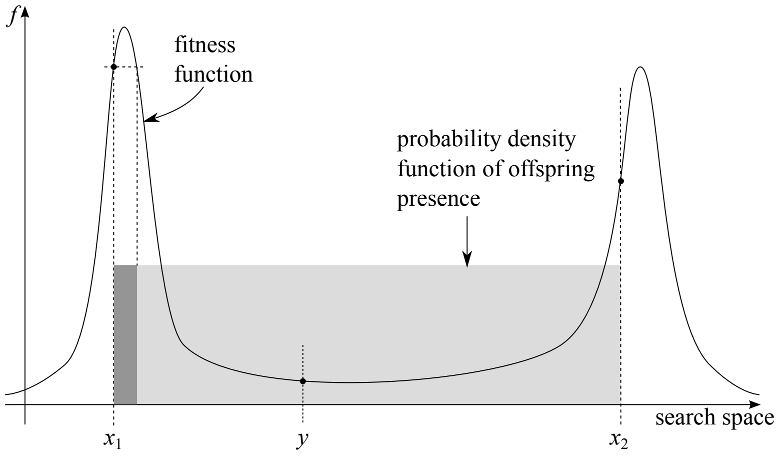


Figure 1.10. *Let x_1 and x_2 be parents placed on different peaks of fitness function f. Assuming that the crossover x_1 and x_2 yields an offspring y uniformly drawn from interval $[x_1, x_2]$, y is likely to have a poor fitness value. The probability that offspring are better than their parents is represented by the dark gray region*

If the fitness function is continuous, the following approaches can be used to avoid a too high proportion of lethal crossovers:

1) by restricting the mating to similar parents;

2) by giving more chance to yield offspring close to their parents, as with the Simulated Binary Crossover [DEB 95].

The first option is inspired by the natural genetics metaphor: individuals of different species cannot be crossed to yield viable offspring if they are too dissimilar. In the frame of evolutionary algorithms, the simplest way consists of crossing individuals if distance between them is less than a threshold r_c called the *restriction radius* [GOL 89]. However, a too small value for r_c could significantly lower the effective crossover rate as well as the exploration in the search space, which could slow down or even freeze the search of the optimum. r_c is difficult to estimate because it depends on distances between peaks, which are not known in general. It is possible to consider a decreasing radius r_c during the evolution to overcome this problem.

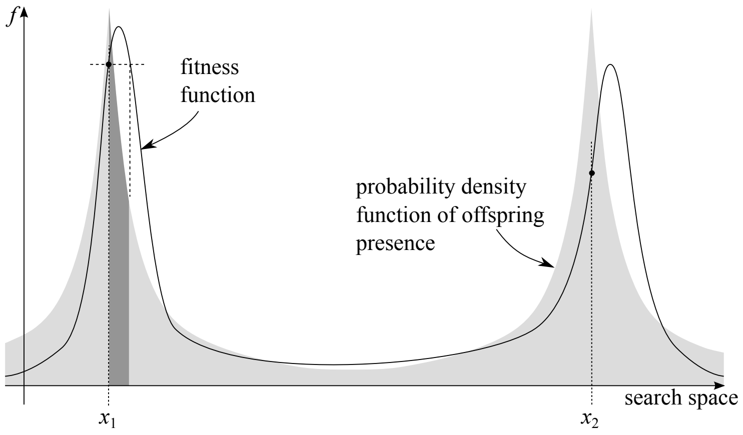


Figure 1.11. *Compared to the case of Figure 1.10, for the same fitness function, a multimodal probability density function of the offspring presence after crossover, for which the modes are parents x_1 and x_2, increases the probability that offspring are better than their parents. This probability is represented by the dark gray region*

Another option to avoid a high proportion of lethal crossover consists of using a multimodal probability distribution of offspring, such that its modes are the parents. An example is given in Figure 1.11. The areas of the dark gray regions in Figures 1.11 and 1.10 represent the probabilities that offspring have better fitness values than those of their parents. By comparing these figures, it appears that this probability is higher with the multimodal distribution.

1.4.3. *Mutation*

Most of the mutation operators alter an individual in such a way that the result of the transformation is often close to it, but not always, to preserve the exploration ability of the operator. They mainly perform a random local search around each individual to mutate. Thus, the mutation gives each individual a chance to approach the exact location of the maximum of a fitness function, as much as the characteristics or the parameters of the chosen operator allow it.

The proportion of the mutated individuals in the offspring population is equal to the *mutation rate*. In the frame of *genetic algorithms* (section 1.6), the mutation is considered as a minor operator, useful to preserve diversity in the population, which the crossover cannot ensure. The mutation rate is then typically low, about 0.01 to 0.1, whereas the crossover rate is high. Conversely, a 100% mutation rate was required for the first *Evolution Strategy* algorithms since they did not use crossover. A large enough mutation rate helps to preserve diversity in the population, which is useful for a good exploration of the search space. Thus, this operator can contribute to fight the reduction of the population variance due to a strong selection pressure or genetic drift.

Using mutation as a local search operator suggests to combine it with other more efficient, although more problem-dependent, local techniques such as a gradient method for example. This approach leads to the design of *hybrid* evolutionary algorithms.

1.5. Binary representation

The idea of evolving a population of binary vectors mainly comes from genetic algorithms, which implements a simple model of the transcription *genotype–phenotype* existing in the living world. Within the framework of genetic algorithms, the genotype of an individual is comprised of a string of binary symbols, or more generally, a string of symbols belonging to a low-cardinality alphabet. The phenotypes are solutions to a problem expressed in a “natural” representation, *a priori* easy to understand by people who want to solve it. They are used by the algorithm only for the fitness evaluation of an individual. Conversely, the binary genotypes are hard to interpret by a human expert. They undergo the action of the genetic operators: mutations and crossover.

For example, if a solution to a given problem is expressed naturally as a vector of real numbers, the phenotype could be this vector. Thus, the genotype is a binary string obtained from this vector. The simplest way consists of converting each of its components with a number of bits corresponding to the required precision. Then, these binary numbers can be concatenated to generate the genotype.

1.5.1. *Crossover*

For a binary representation, there exist three widespread variants of crossovers:

– the “single point” crossover;

– the “two point” crossover;

– the uniform crossover.

A pair of individuals being chosen randomly among the selected individuals, the “single point” crossover [HOL 92] is applied in two steps:

1) random choice of an identical cut point on the two bit strings (see Figure 1.12(a));

2) cut of the two strings (Figure 1.12(b)) and exchange of the two fragments located on the right (Figure 1.12(c)).

This operator generates two offspring from two parents. If only one offspring is needed by the evolutionary algorithm, it is chosen at random in the pair and the other one is discarded. The “single point” crossover is the simplest one for codings with a low cardinality alphabet, like binary coding.

A “two point” crossover can be implemented by randomly choosing two cut points c_1 and c_2, with $c_1 < c_2$, in parent individuals as shown in Figure 1.13. Symbols whose indices are in interval $[c_1, c_2]$ are exchanged between the two strings to obtain an offspring.

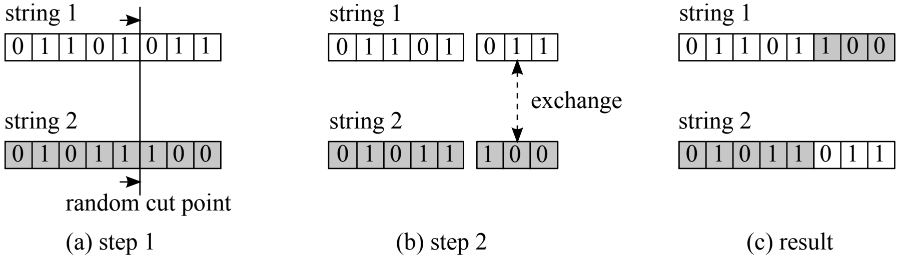


Figure 1.12. *"Single point" crossover of two 8 bit strings*

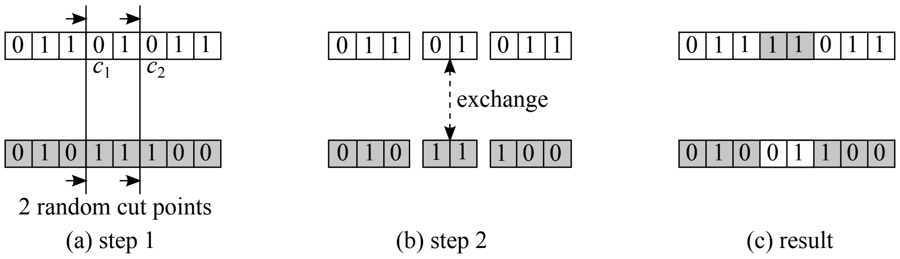


Figure 1.13. *"Two point" crossover of two 8 bit strings*

The "single point" and "two point" crossovers are usually employed in practice for their simplicity and efficiency. An immediate generalization of these operators consists of multiplying the cut points on each string. The uniform crossover [ACK 87] can be viewed as a multipoint crossover where the number of cuts is *a priori* unspecified. Practically, a "template string" is used. It is a binary string of the same length as the individuals. A "0" at the n^{ith} position of the template leaves the symbols in the n^{ith} position of the two strings unchanged. A "1" activates an exchange of the corresponding symbols (in Figure 1.14). The template is generated at random for each pair of individuals. The values "0" or "1" of the template elements are generally drawn with a probability of 0.5. Lower probabilities generate offspring closer to the parents in the sense of the Hamming distance[3].

3 Hamming distance: number of different symbols between two strings of the same length.

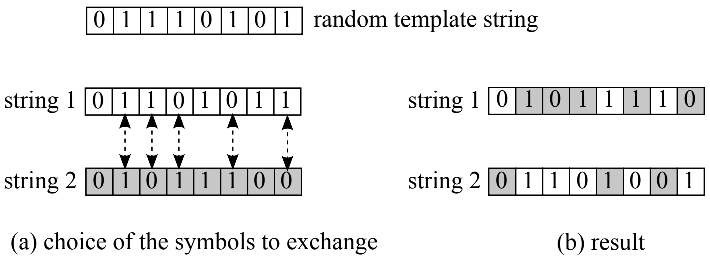


Figure 1.14. *Uniform crossover*

1.5.2. *Mutation*

The mutation operator on bit strings changes some of its symbols at random. The most common variants are the *deterministic mutation* and the *bit-flip mutation*. With the “deterministic” mutation, a fixed number of bits chosen at random are reversed for each mutated individual, i.e. a “1” becomes “0” and vice versa. With the “bit-flip” mutation, each bit can be reversed independently of the others according to a given probability. These operators have good exploitation abilities if the number of reversed bits is small enough in accordance with the Hamming distance (see section 1.4.1). However, this number should also be large enough to preserve the diversity in the population. This parameter is problem-dependent.

When a bit string represents a vector of integer or real numbers, the search for the optimum of the fitness function might be countered by the difficulty of crossing the *Hamming cliffs*, due to the conversion of the bit strings towards real number vectors. For example, let us consider function $C(x)$ defined below:

$$C(x) = \begin{cases} x & \text{if } x \leq 16 \\ 0 & \text{otherwise} \end{cases}$$

Let $b(x) = \{b_1(x), \ldots, b_5(x)\}$ be a string of five bits to represent an integer individual x that ranges from 0 to 31. $b(x)$ can be simply defined as the standard representation of x in base 2. The maximum of $C(x)$ is obtained for $x = 16$, which thus corresponds to $b(0) = \{1, 0, 0, 0, 0\}$. The value $x = 15$, obtained from the string $\{0, 1, 1, 1, 1\}$, yields the highest fitness apart from the maximum: this value will thus be favored by the selection operators.

However, there is no common bit between $\{1, 0, 0, 0, 0\}$ and $\{0, 1, 1, 1, 1\}$. This means that there is no other individual with which $\{0, 1, 1, 1, 1\}$ can be crossed to give $\{1, 0, 0, 0, 0\}$. As for the mutation operator, it will have to change all the bits of string $\{0, 1, 1, 1, 1\}$ simultaneously to obtain the optimum. The Hamming distance between the optimum and the individual which has the nearest fitness is maximal, equal to the length of the strings. This is a Hamming cliff. It is unlikely to jump over it with a "bit-flip" mutation, and it is impossible with the "deterministic" mutation, unless it flips all the bits of the string, which is never used.

But the mutation will be able to easily find the optimum if there are individuals in the population that differ in only one bit of the optimal string. Here, these individuals are:

String $b(x)$	x	$C(x)$
$\langle 0, 0, 0, 0, 0\rangle$	0	0
$\langle 1, 1, 0, 0, 0\rangle$	24	0
$\langle 1, 0, 1, 0, 0\rangle$	20	0
$\langle 1, 0, 0, 1, 0\rangle$	18	0
$\langle 1, 0, 0, 0, 1\rangle$	17	0

Unfortunately, all these individuals have zero fitness and thus they are not likely to "survive" from one generation to the next one.

This annoying phenomenon, which hinders the progress towards the optimum, can be eliminated by choosing a *Gray code*, which ensures that two successive integers will have binary representations that differ only in one bit. Starting from strings $b(x)$ that represent integer numbers in base 2, it is easy to obtain a Gray code $g(x) = \{g_1(x), \ldots, g_l(x)\}$ by performing, for each bit i, the operation:

$$g_i(x) = b_i(x) \oplus b_{i-1}(x)$$

where the operator $\oplus$ implements the "exclusive or" operation and $b_0(x) = 0$. Conversely, the string of l bits $b(x) = \{b_1(x), \ldots, b_l(x)\}$ can be obtained from the string $g(x) = \{g_1(x), \ldots, g_l(x)\}$:

$$b_i(x) = \bigoplus_{j=1}^{i} g_j(x)$$

The Gray codes of $\{0,1,1,1,1\}$ and $\{1,0,0,0,0\}$ are respectively $\{0,1,0,0,0\}$ and $\{1,1,0,0,0\}$. The mutation of bit g_1 is therefore enough to reach the optimum. A Gray code is thus desirable from this point of view.

However, the Hamming cliffs are generally not the cause of significant decrease in the performance of a search algorithm. More details about Gray codes and binary representations are developed in [ROW 04]. Especially, it is noted that the use of several Gray codes modifies the landscape of the fitness function and can help to escape from local optima.

1.6. The simple genetic algorithm

The simple genetic algorithm is an evolutionary algorithm, similar to the one presented in Figure 1.1 (p. 4), with a notable particularity: it implements a transcription *genotype–phenotype* inspired by natural genetics. The *phenotype* is the expression of a solution to a problem in its usual formalism. A *genotype* is a string of symbols (often binary) from which the associated phenotype is built. The phenotype can then be evaluated to give a fitness value that can be used by the selection operators.

The flowchart of a simple genetic algorithm is presented in Figure 1.15. It implements a proportional selection operator (see section 1.3.3, p. 9) and a generational replacement, i.e. the population of the offspring replaces that of the parents. Another classical variant uses the steady state replacement (section 1.3.6.5, p. 19).

The variation operators alter the genotypes. As they are bit strings, crossover and mutation operators for binary strings (section 1.5) are naturally used. The crossover is considered as the main variation operator. The mutation is usually applied with a small rate to maintain diversity in the population [GOL 89]. An appropriate coding of the genotype should be designed, such that the variation operators produce viable offspring, satisfying the constraints of the problem as much as possible.

Holland, Goldberg and many other authors have worked on a mathematical formalization of the genetic algorithms based on a “Schema Theorem” [GOL 89]. It provides arguments for the choice of a binary representation. However, deceiving results obtained with this theorem have given rise to controversies and debates about its utility [VOS 98]. In particular, the suitability of binary encoding has been challenged.

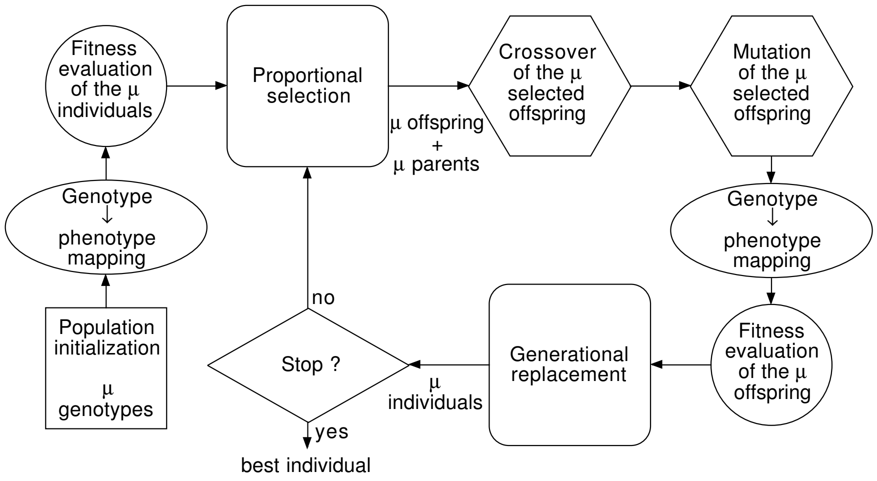

Fitness evaluation of the μ individuals
Proportional selection
μ offspring + μ parents
Crossover of the μ selected offspring
Mutation of the μ selected offspring
Genotype ↓ phenotype mapping
Genotype ↓ phenotype mapping
Population initialization μ genotypes
no
Stop ?
yes
best individual
μ individuals
Generational replacement
Fitness evaluation of the μ offspring

Figure 1.15. *A simple genetic algorithm*

Many variants of genetic algorithms have been proposed in order to improve their performances or to extend their application domains. Thus, the bit strings have been replaced by other data structures closer to the natural formalism of the problems to solve, provided that appropriate variation operators are available. This avoids the difficult and complex question of the design of an efficient coding. For example, the “Real Coded Genetic Algorithms” use genotypes that are real vectors, instead of binary strings, to solve problems defined in $\mathbb{R}^n$. In addition, proportional selection is often replaced by other kinds of selection. These modifications are significant enough that the specific features of the genetic algorithms blend in with the diversity of the other evolutionary approaches.

1.7. Conclusion

This introductory chapter has presented the basic principles of evolutionary algorithms and a collection of widespread selection and variation operators. Binary representation has been addressed to introduce the transcription genotype–phenotype implemented in the genetic algorithm. Other representations are used in the world of evolutionary algorithms. They will be described in further chapters with their associated variation operators.

2

Continuous Optimization

2.1. Introduction

In the sixties, Ingo Rechenberg and Hans Paul Schwefel worked on the optimization of device shapes related to fluid dynamics, such as wing profiles. They proceeded experimentally with a “generate and test” approach. From this process, they formalized the first *Evolution Strategy* (ES) algorithm [REC 73] [BAE 91]. The method aims to find a global optimum in continuous domains without requiring differentiability or continuity of the objective functions. Such features are useful when objective functions are rugged or noisy.

Over the years, evolution strategies have improved in regard to several aspects. For example, self-adaptation of the mutation parameters was introduced [REC 73] to reach the optimum with a high accuracy while increasing the convergence speed. Some such mechanisms for ES are described in section 2.2.2. Self-adaptation has become one of the major research axes for the evolutionary algorithms of today. Indeed, it is not acceptable to have to search good values of possibly numerous and abstruse critical parameters of an optimizer, before being able to use it to solve a given class of optimization problems efficiently [EIB 11] [KAR 15].

The first attempts to solve continuous optimization problems with genetic algorithms were based on binary coding of real numbers [GOL 89]. However, two different closest neighboring points in a binary vector space $\{0, 1\}^n$ may correspond to distant points in the space generated by the decoding of these binary vectors and vice versa. Thus, binary coding generates some drawbacks as explained in section 1.5.2. For this reason, authors have proposed mutation and crossover operators that are able to directly deal with vectors of real numbers, without transformation from one space to another. Some of such basic, non-self-adaptive operators among the most popular, are described in section 2.2 of this chapter.

These variation operators for the *Real Coded Genetic Algorithms* (RCGA) have the ability to escape local optima of massively multimodal functions, such as the Rastrigin function, which has 11^d local minima in domain $[-5.12, 5.12]^d$ in dimension d and a global minimum value of zero (see section 2.10). For instance, experiments in dimension $d = 10$ show that a RCGA with

- a population of $\lambda = \mu = 5000$ individuals,
- SUS parental selection (section 1.3.3.2, p. 11),
- isotropic Gaussian mutation with $\sigma = 1$ (section 2.2),
- uniform crossover (section 2.2) and
- $(\lambda + \mu)$ elitist replacement (section 1.3.6.3, p. 18)

achieves a median objective value of 0.06 after 10^6 evaluations of the objective function, for each of the 100 runs. The value 0.06 indicates a quite good performance. However, when the same optimizer is applied to randomly rotated Rastrigin functions, the median objective value after 10^6 evaluations degrades to 3.7! In fact, most of the operators presented in section 2.2 are efficient for well-conditioned separable objective functions (see section 2.2.2.4). But randomly rotated Rastrigin functions are non-separable. Ill-conditioned objective functions also cause significant performance losses with the RCGA described above.

To overcome these problems, researchers have proposed new algorithms to increase the search efficiency of global optima for multimodal, ill-conditioned and non-separable objective functions. This chapter presents three leading evolutionary approaches of today. The canonical *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES), introduced in section 2.3, is a major milestone for the Evolution Strategies. To improve the performance of CMA-ES on multimodal functions, a restart strategy has been proposed referred to as the IPOP-CMA-ES, described in section 2.4. Taking another point of view, *Differential Evolution* (DE) algorithms used an original simple self-adaptive mutation operator that is able to deal with ill-conditioned, non-separable functions. Section 2.5 is devoted to the canonical DE algorithm, from which many variants have been proposed. Among them, section 2.6 presents the powerful *Success-History based Adaptive Differential Evolution* (SHADE) algorithm. The Particle Swarm Optimization approaches are very popular metaheuristics. Section 2.7 presents a high performance variant: the Standard Particle Swarm Optimizer 2007 (SPSO-2007), which is close to the original algorithm. Thus, when practitioners plan to choose an evolutionary optimizer, they are faced with a variety of methods. Section 2.8 aims to discuss the performance of the algorithms presented in this chapter according to several criteria on a set of classical test objective functions.

2.2. Real representation and variation operators for evolutionary algorithms

Let us assume that any solution $\mathbf{x}$ of a given population of offspring is drawn from the search domain according to a probability distribution characterized by a density $p(\mathbf{x})$, where $\mathbf{x}$ is a point in $\boldsymbol{\Omega}$. Moreover, it is assumed that this distribution has an expectation:

$$E = \int_{\boldsymbol{\Omega}} \mathbf{x} p(\mathbf{x}) d\mathbf{x}$$

and a total variance:

$$V = \int_{\boldsymbol{\Omega}} \mathbf{x}^2 p(\mathbf{x}) d\mathbf{x} - E^2$$

V is also the trace of the covariance matrix of the components of vectors $\mathbf{x}$. If λ, which is the size of the population of the offspring, is large enough, these values are approached by the empirical expectation:

$$\hat{E} = \frac{\sum_{i=1}^{\lambda} \mathbf{x}_i}{\lambda}$$

and the empirical total variance:

$$\hat{V} = \frac{\sum_{i=1}^{\lambda} \mathbf{x}_i^2}{\lambda} - \hat{E}^2$$

Empirical variance can be regarded as a measurement of diversity in the population. If it is zero, then all the individuals are at the same point in $\mathbf{\Omega}$. It is interesting to evaluate these values after application of the variation operators.

2.2.1. *Crossover*

Let us consider two points $\mathbf{x}$ and $\mathbf{y}$ in space $\mathbb{R}^n$ corresponding to two individuals selected to generate offspring. After the application of the crossover operator, one or two offspring $\mathbf{x}'$ and $\mathbf{y}'$ are chosen randomly, according to a probability distribution which depends on $\mathbf{x}$ and $\mathbf{y}$.

2.2.1.1. *Crossover by exchange of components*

It is a direct generalization of the binary crossovers, which consists of exchanging some real components of two parents. It is thus possible to return to the variants of the binary crossover, in particular the "single point", "two point" and "uniform" crossovers (see Figure 2.1). The last variant is also called "discrete recombination" according to the terminology of the *Evolution Strategies*. This kind of crossover modifies neither E nor V.

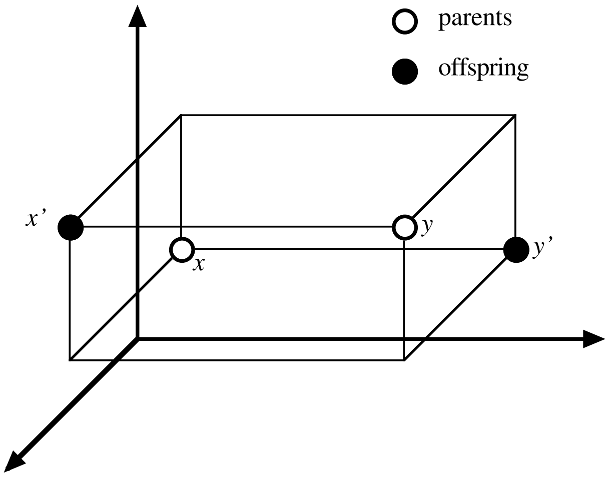


Figure 2.1. *Uniform crossover; an individual x' or y' resulting from the crossover of x and y is located on a vertex of a hyper-rectangle with sides parallel to the coordinate axes such that a longest diagonal is the segment (x, y)*

2.2.1.2. *BLX-α crossover*

The *BLX-α crossover* was proposed by [ESH 93], α being a parameter of the evolutionary algorithm. Two variants are widely mentioned in publications related to evolutionary algorithms. According to the original description of the authors, the first one randomly generates offspring on a line segment in a search space $\mathbb{R}^n$ passing through the two parents. We refer to this variant as the *linear BLX-α crossover*. The second variant randomly generates offspring inside a hyper-rectangle defined by the parents. We designate it as the *voluminal BLX-α crossover*.

2.2.1.2.1. Voluminal BLX-α crossover

This operator generates an offspring $\mathbf{z}$ chosen uniformly inside a hyper-rectangle with sides parallel to the coordinate axes (see Figure 2.2). Let x_i and y_i be the components of the two parents $\mathbf{x}$ and $\mathbf{y}$ respectively, for $1 \le i \le n$; the components of an offspring $\mathbf{z}$ are defined as:

$$z_i = x_i + (y_i - x_i) \cdot \mathcal{U}(-\alpha, 1 + \alpha)$$

where $\mathcal{U}[-\alpha, 1 + \alpha]$ is a random number drawn uniformly in the interval $[-\alpha, 1 + \alpha]$. In this way: $z_i \in [x_i - \alpha(y_i - x_i), y_i + \alpha(y_i - x_i)]$.

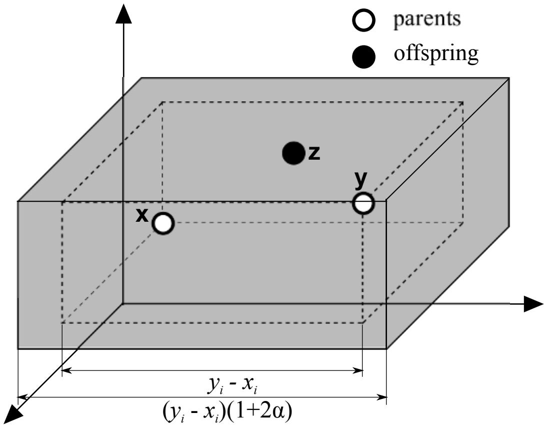


Figure 2.2. *BLX-α crossover; an individual $\mathbf{z}$ resulting from the crossover of $\mathbf{x}$ and $\mathbf{y}$ is located inside a hyper-rectangle with sides parallel to the coordinate axes such that a longest diagonal passes through $\mathbf{x}$ and $\mathbf{y}$*

The voluminal BLX-α crossover does not modify E, but changes the value of V. Let V_c be the variance of distribution of the population after crossover:

$$V_c = \frac{(1+2\alpha)^2+3}{6}V$$

The variance after crossover decreases if:

$$\alpha < \frac{\sqrt{3}-1}{2} \approx 0.366$$

In this case, it is said that the crossover is *contracting*, and the iterative application of the operator alone leads the population to collapse on its centroid. In particular, if $\alpha = 0$, z is located in the hyper-rectangle such that a longest diagonal is the line segment (x, y). In this case, $V_c = \frac{2}{3}V$. After iterative application of this operator alone for g generations, and for an initial population variance V_0, the variance becomes:

$$V_{cg} = \left(\frac{2}{3}\right)^g V_0$$

The variance tends quickly to 0! It is thus seen that the risk of premature convergence is increased with a BLX-0 operator.

If $\alpha > \frac{\sqrt{3}-1}{2}$, the variance increases if the domain is $\mathbb{R}^n$. In practice, for a bounded search domain $\boldsymbol{\Omega}$, the variance is stabilized with a non-zero value. The "boundaries" of the search domain can be explored. The possible optima which are there will be more easily found and retained. A usual value is $\alpha = 0.5$.

Nomura and Shimohara [NOM 01] have shown that the operator reduces the possible correlations which exist between the components of the vectors of the population. Its repeated application makes the coefficients of correlation converge towards zero.

This operator can be seen as a generalization of other crossover operators, such as the *flat crossover* [RAD 90] which is equivalent to BLX-0.

2.2.1.2.2. Linear BLX-α crossover

$\mathbf{x}$ and $\mathbf{y}$ being the points corresponding to two individuals in $\mathbb{R}^n$, an individual $\mathbf{z}$ resulting from the linear BLX-α crossover of $\mathbf{x}$ and $\mathbf{y}$ is chosen according to a uniform distribution on a line segment passing through $\mathbf{x}$ and $\mathbf{y}$:

$$\mathbf{z} = \mathbf{x} + (\mathbf{y} - \mathbf{x}) \cdot \mathcal{U}(-\alpha, 1 + \alpha)$$

where $\mathcal{U}(-\alpha, 1 + \alpha)$ is a random number uniformly drawn from the interval $[-\alpha, 1 + \alpha]$. If I is the length of the line segment $[\mathbf{x}, \mathbf{y}]$, $\mathbf{z}$ is on the segment of length $I \cdot (1 + 2\alpha)$ centered on the segment $[\mathbf{x}, \mathbf{y}]$ (Figure 2.3).

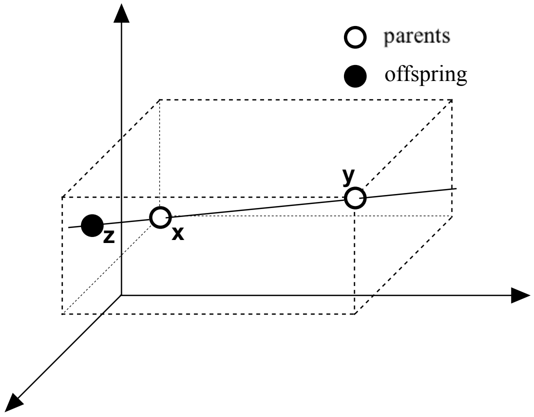


Figure 2.3. *BLX-α crossover; an individual $\mathbf{z}$ resulting from the crossover of $\mathbf{x}$ and $\mathbf{y}$ is located on the line defined by $\mathbf{x}$ and $\mathbf{y}$, possibly outside the segment $[\mathbf{x}, \mathbf{y}]$*

The linear BLX-α crossover does not modify E, but changes the value of V in a way similar to the voluminal BLX-α crossover. On the other hand, it is noted that the possible correlations existing between the components of the individuals of a population are not decreasing by the repeated application of the linear operator [NOM 01]. This behavior is completely different from that observed for the voluminal operator.

This operator can be seen as a generalization of other crossover operators, according to restrictions on the values of α, like the *intermediate recombination* for the "Evolution Strategies" [BEY 02] or the *arithmetic crossover* [MIC 96a], which is equivalent to BLX-0.

2.2.1.3. *Intermediate recombination*

This operator is applied to ρ parents and gives one offspring each time it is invoked. ρ is a constant parameter between 2 and the population size. An offspring $\mathbf{z}$ is the centroid of parents $\mathbf{x}_i$:

$$\mathbf{z} = \frac{1}{\rho}\sum_{i=1}^{\rho}\mathbf{x}_i$$

2.2.2. *Mutation*

The mutation generally consists of the addition of a “small” random value to each component of an individual, according to a zero average distribution, with a variance possibly decreasing with time. In this way, it is assured that the mutation leaves the centroid of the population unchanged.

2.2.2.1. *Uniform mutation*

The simplest mutation technique adds to an individual x, belonging to a domain $\boldsymbol{\Omega}$ in $\mathbb{R}^n$, a random variable of uniform distribution in a hyper-cube $[-a, +a]^n$. However, such a mutation does not allow an individual trapped in a local optimum located on a peak broader than the hypercube to escape from it. To avoid this disadvantage, it is preferable to use an unlimited support distribution.

2.2.2.2. *Gaussian mutation*

The Gaussian mutation is one of the most widely used for the real representation. The simplest form adds a Gaussian random variable $\mathcal{N}(0, \sigma)$, of zero average and standard deviation σ to each component of a real valued vector. The problem is then: how to choose an adequate value for σ? In theory, it is possible to escape from a local optimum irrespective of the width of the peak where it is, since the support of a Gaussian distribution is unlimited, but if σ is too small that could happen after far too many attempts. Conversely, if σ is too large, it will be unlikely to approach an optimum value accurately within a reasonable time. The value of σ should be then adapted during the evolution: large at the beginning to quickly explore the search space, small at the end to accurately approach the optimum. Some adaptation strategies are described in the following.

2.2.2.3. *Gaussian mutation and the* $1/5$ *rule*

According to a study on two simple and very different test functions with an elitist Evolution Strategy $(1 + 1)-$ES[1], Rechenberg [REC 73] [BEY 01] calculated optimal standard deviations for each test function that maximize the convergence speed. He observed that for these optimal values, approximately one-fifth of the mutations allow to reduce the distance between the individual and the optimum. He deduced the following rule, termed as the "one fifth rule" to adapt σ: *if the rate of the successful mutations is larger than* $1/5$*, increase* σ*, if it is smaller, reduce* σ. The "rate of the successful mutations" is the proportion of mutations which make it possible to improve the value of fitness of an individual. Schwefel [SCH 81] proposed the following rule in practice:

```
Estimate the rates of successful mutations $p_s$ on $k$ mutations
**if** $p_s < 0.2$ **then**
  $\sigma(g) \leftarrow \sigma(g) \cdot a$
**else if** $p_s > 0.2$ **then**
  $\sigma(g) \leftarrow \sigma(g)/a$
**else**
  $\sigma(g) \leftarrow \sigma(g)$
```

with $0.85 \leq a < 1$ according to the recommendation of Schwefel. k is equal to dimension n of the search space when $n > 30$ and g is the index of the current generation. σ should be updated according to the above algorithm for every n mutations.

2.2.2.4. *Remarks on condition number and separability of objective functions*

The "one fifth rule" requires that σ should have the same value for all the components of a vector x. In this way, the progression step towards the optimum is the same in all directions: the mutation is isotropic. However, the isotropy does not make it possible to approach the optimum as quickly as expected when, for example, the iso-values of the fitness function locally take the shape of "flattened" ellipsoids in the neighborhood of the optimum (see Figure 2.4). If the step is well adapted in a particular direction, it will not be in the other directions.

1 (1 + 1)-ES: the population is composed of only one parent individual that generates only one offspring, the best of both is preserved for the next generation.

To clarify these considerations with an example, we consider the quadratic performance function defined in $\mathbb{R}^n$: $f(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{c})^{\mathrm{T}}\mathbf{H}(\mathbf{x} - \mathbf{c})$, where $\mathbf{H}$ is a symmetric matrix. This example is interesting because the expression of $f(\mathbf{x})$ is the second-order term of a Taylor expansion near the point $\mathbf{c}$ for any function twice continuously differentiable, where $\mathbf{H}$ is the Hessian matrix of this function at $\mathbf{c}$. $f(\mathbf{x})$ is minimal, equal to 0, for $\mathbf{x} = \mathbf{c}$ with $\mathbf{H}$ positive definite. Figure 2.4 represents the iso-value ellipse $f(x_1, x_2) = 1/2$ for a function of two variables obtained when $\mathbf{H}$ is diagonal with $h_{11} = 1/36$ and $h_{22} = 1$.

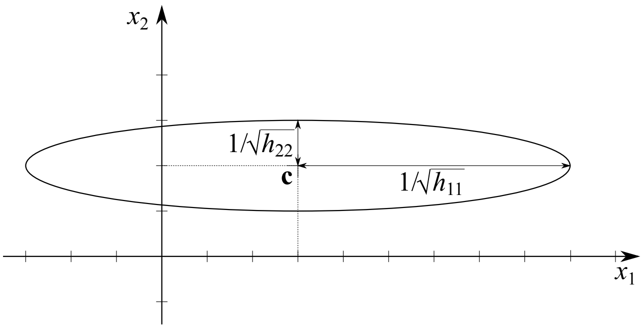


Figure 2.4. *Iso-value ellipse $f(x_1, x_2) = 1/2$ when $\mathbf{H}$ is diagonal with $h_{11} = 1/36$ and $h_{22} = 1$*

The condition number $\kappa_{\mathbf{H}}$ is defined as the ratio of the largest eigenvalue of $\mathbf{H}$ over the smallest one: $\kappa_{\mathbf{H}} = \lambda_{\max}/\lambda_{\min}$. In the case shown in Figure 2.4, the matrix $\mathbf{H}$ is already diagonal, its eigenvalues are h_{11} and h_{22}. For $\mathbf{H}$ defined above, the condition number is 36. When the condition number is large compared to 1, the matrix is said to be "*ill-conditioned*". For real-world applications, the condition number can be larger than 10^{10}, which means that the length ratio between the major axis and the minor axis of an iso-value hyper-ellipsoid can be larger than 10^5.

Note that, when $\mathbf{H}$ is a diagonal, the quadratic function $f(\mathbf{x})$ is an additively separable function: $f(\mathbf{x}) = f(x_1, ..., x_i, ..., x_n) = \sum_{i=1}^{n} g(x_i)$. Thus, when $\mathbf{H}$ is positive definite, the global minimum of $f(\mathbf{x})$ can be obtained by searching the minima of n convex functions $f_i(x_i) = f(c_1, ..., c_{i-1}, x_i, c_{i+1}, ..., c_n)$ with constants $c_1, ..., c_{i-1}, c_{i+1}, ..., c_n$

arbitrarily set. In this case, the optimum of $f(\mathbf{x})$ can be efficiently found with n successive runs of the (1+1)-ES algorithm with the "1/5 rule" to obtain the optimum for each of variables x_i independently of the others. $\mathbf{H}$ being diagonal, the ratios of the adapted standard deviations σ_i/σ_j at a given generation are ideally of the order of $\sqrt{h_{jj}}/\sqrt{h_{ii}}$ to reduce the computation time at best. However, such an approach to solve ill-conditioned problems cannot be applied efficiently when the objective function is not separable.

2.2.2.5. *Self-adaptive Gaussian mutation*

Schwefel [SCH 81] has proposed the *self-adaptive Gaussian mutation* to efficiently solve ill-conditioned problems when the objective function is separable or "almost" separable in a neighborhood of the optimum. The self-adaptive mutation should be applicable to a wider range of applications than the "1/5 rule" because the latter was derived from the study of specific objective functions [BEY 01].

To implement this adaptation, an individual is represented as a pair of vectors $(\mathbf{x}, \sigma)$. Each component σ_i refers to the corresponding component of $\mathbf{x}$. σ_i evolves in a similar way to the variables of the problem under the action of the evolutionary algorithm [SCH 81]. σ is thus likely to undergo mutations. Schwefel has proposed that the couple (x', σ'), obtained after mutation, is such that:

$$\sigma'_i = \sigma_i \exp(\tau_0 N + \tau \mathcal{N}(0,1)) \qquad [2.1]$$

$$\text{with} \qquad \tau_0 \approx \frac{1}{\sqrt{2n}} \qquad \tau \approx \frac{1}{\sqrt{2\sqrt{n}}}$$

$$x'_i = x_i + \mathcal{N}(0, {\sigma'_i}^2)$$

where N indicates a Gaussian random value of average 0 and variance 1, computed for the entire set of n components of σ, and $\mathcal{N}(0, v)$ represents a Gaussian random variable of average 0 and variance v. σ'_i is thus updated by application of a lognormal perturbation (equation [2.1]).

The self-adaptive Gaussian mutation requires a population size μ of the order of the search space dimension. [BEY 02] recommend to associate this mutation with the intermediate recombination (section 2.2.1.3) to prevent excessive fluctuations of the parameters that degrade the performance of the algorithm. As for the "1/5 rule", this operator becomes inefficient when the objective function is ill-conditioned and not separable in the neighborhood of the optimum.

2.2.2.6. *Correlated Gaussian mutation*

The self-adaptive mutation described above works best when the matrix $\mathbf{H}$ is diagonal. It is inefficient when there are correlations between variables, as in the case of the fitness function where the iso-value curve $f(\mathbf{x}) = 1/2$ is represented in Figure 2.5. This case corresponds to a matrix $\mathbf{H} = (\mathbf{DR})^\mathsf{T} (\mathbf{DR})$ where $\mathbf{D}$ is the diagonal matrix of the square roots of the eigenvalues of $\mathbf{H}$ and $\mathbf{R}$ is a rotation matrix with:

$$\mathbf{D} = \begin{pmatrix} 1/6 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } \mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \text{ with } \theta = \pi/6 \qquad [2.2]$$

The condition number $\kappa_\mathbf{H} = (s_{22}/s_{11})^2$ is then equal to 36. This function f for which there are correlations between variables is not separable.

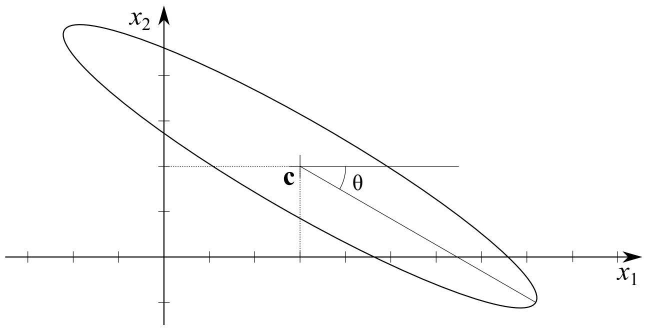


Figure 2.5. *An iso-value curve $f(\mathbf{x}) = 1/2$ with $f(\mathbf{x}) = \frac{1}{2}(\mathbf{x}-\mathbf{c})^\mathrm{T}\mathbf{H}(\mathbf{x}-\mathbf{c})$ obtained for $\mathbf{H} = (\mathbf{DR})^\mathsf{T} (\mathbf{DR})$ where $\mathbf{D}$ and $\mathbf{R}$ are given by expressions [2.2]*

The correlated mutation is a generalization of the self-adaptive mutation described above. The mutated vector $\mathbf{x}'$ is obtained from $\mathbf{x}$ by the addition of a Gaussian random vector with zero mean and covariance matrix $\mathbf{C}$:

$$\mathbf{x}' = \mathbf{x} + \mathcal{N}(0, \mathbf{C})$$

The matrix $\mathbf{C}$, which is symmetric positive definite, can always be written as $\mathbf{C} = (\mathbf{SR})^{\mathrm{T}}(\mathbf{SR})$ where $\mathbf{R}$ is a rotation matrix in $\mathbb{R}^n$ and $\mathbf{S}$ is a diagonal matrix with $s_{ii} > 0$ [RUD 92][2]. The matrix $\mathbf{R}$ can be computed as the product of $n(n-1)/2$ elementary rotation matrices $\mathbf{R}_{kl}(\alpha_{kl})$:

$$\mathbf{R} = \prod_{k=1}^{n-1} \prod_{l=k+1}^{n} \mathbf{R}_{kl}(\alpha_{kl})$$

$\mathbf{R}_{kl}(\alpha_{kl})$ is the rotation matrix whose angle is α_{kl}, in the plane spanned by vectors k and l. Such a matrix is written as the identity matrix at the exception of coefficients $r_{kk} = r_{ll} = \cos(\alpha_{kl})$ and $r_{kl} = -r_{lk} = -\sin(\alpha_{kl})$.

Each individual has its own covariance matrix $\mathbf{C}$. It is able to adapt itself through mutations of the information required to build it. Thus, an individual can be defined as a triplet $(\mathbf{x}, \sigma, \alpha)$, where σ is a vector of n standard deviations, as for the self-adaptive mutation, and α is a vector *a priori* composed of $n(n-1)/2$ elementary rotation angles α_{kl} used to build the matrix $\mathbf{R}$. The diagonal coefficients of the matrix $\mathbf{S}$ are $s_{ii} = \sigma_i$.

σ evolves under the action of the evolutionary algorithm as described by equation [2.1]. Components α_{kl} undergo mutations according to the following formula:

$$\alpha'_{kl} = \alpha_{kl} + \beta\mathcal{N}(0,1)$$

Schwefel suggests to set β at a value close to 0.087 radian, i.e. 5 degrees.

In practice, the mutated vector $\mathbf{x}'$ is obtained from $\mathbf{x}$ according to the following expression:

$$\mathbf{x}' = \mathbf{x} + \mathbf{R}'\mathbf{S}'\mathcal{N}(0,\mathbf{I})$$

$\mathbf{R}'$ and $\mathbf{S}'$ are respectively obtained from $\mathbf{R}$ and $\mathbf{S}$ after mutation of angles α_{kl} and standard deviations σ_i. $\mathcal{N}(0,\mathbf{I})$ is a Gaussian random vector with zero mean and variance 1 for each component i.

2 Possibly with a column permutation of the matrix $\mathbf{R}$ and the corresponding diagonal coefficients in $\mathbf{S}$.

This technique of mutation, although it seems powerful, is seldom used because of the amount of memory used by an individual and its algorithmic complexity of the order of n^2 matrix products for a problem of n variables for each generation. Moreover, the large number of parameters required for each individual involves a large population size of the order of n^2. The method loses much efficiency when dimension n increases. It is hardly possible to exceed dimension 10 [HAN 06].

The difficulties in the use of the correlated mutation method have prompted the search for new approaches leading to a major improvement of Evolution Strategies known as the "*Covariance Matrix Adaptation Evolution Strategy*" (CMA-ES), which is presented below.

2.3. Covariance Matrix Adaptation Evolution Strategy

2.3.1. *Method presentation*

The "Covariance Matrix Adaptation Evolution Strategy" (CMA-ES) was first proposed by Hansen and Ostermeier in 1996 [HAN 96]. It was originally designed to find the global optimum of an objective function in a continuous space such as $\mathbb{R}^n$ with a greater efficiency than Evolution Strategies using the correlated mutation (p. 44). In a similar way, the method performs an evolution by building a sample of λ solutions for each generation g. They are generated randomly according to the Gaussian distribution $\mathcal{N}(\mathbf{m}(g), \mathbf{C}(g))$, with mean vector $\mathbf{m}(g)$ and covariance matrix $\mathbf{C}(g)$). However, unlike the previous Evolution Strategies, the μ best solutions are then selected in this sample to estimate a new Gaussian distribution $\mathcal{N}(\mathbf{m}(g+1), \mathbf{C}(g+1))$, which will be then used at the next generation. There is no more "individual" dependency between some parents and their offspring. The distribution $\mathcal{N}(\mathbf{m}(g+1), \mathbf{C}(g+1))$ is constructed to approach (hopefully enough) the desired optimum. As for the other Evolution Strategies (p. 43 onwards), the CMA-ES algorithms implement a concept of parameter self-adaptation.

In the CMA-ES approach, three parameters $\mathbf{m}$, σ and $\mathbf{C}'$ are considered to define the Gaussian distribution $\mathcal{N}(\mathbf{m}, \sigma^2\mathbf{C}')$, where $\sigma \in \mathbb{R}^+$ is the step size. This decomposition of the covariance matrix $\mathbf{C}$ in two terms makes it possible to separately adjust parameters σ and $\mathbf{C}'$ according to different criteria in order to speed up the convergence towards the optimum [HAN 16]. The following sections describe the selection step of the best solutions generated at random and the adaptation mechanisms of $\mathbf{m}$, σ and $\mathbf{C}'$.

2.3.1.1. *Fitness function and selection*

At the beginning of generation g, λ solutions $\mathbf{x}_i(g) \in \mathbb{R}^n$ are generated randomly according to the Gaussian distribution $\mathcal{N}(\mathbf{m}(g), \sigma(g)^2\mathbf{C}'(g))$. Rank i is assigned to solution $\mathbf{x}_i$ such that the objective value $F(\mathbf{x}_i)$ is better than or equal to $F(\mathbf{x}_{i+1})$ for all i. "Better" means "smaller" for a minimization problem or "larger" for a maximization problem. Solution $\mathbf{x}_i$, for $i \in \{1, ..., \mu\}$, is associated with fitness values f_i decreasing with index i: $\forall i \in \{1, ..., \mu\}, f_i > 0, f_i \geq f_{i+1}$, with $\sum_{i=1}^{\mu} f_i = 1$. f_i values depend only on rank i and are constant throughout evolution. The easiest way is to choose $f_i = 1/\mu$. More sophisticated fitness functions may improve the convergence to the optimum.

The selection is deterministic: it keeps the μ best solutions, which are $\mathbf{x}_1(g)$ to $\mathbf{x}_\mu(g)$.

2.3.1.2. *Adaptation of "m"*

The value of $\mathbf{m}(g+1)$ for the next generation is the average weighted by the fitness values f_i of the μ selected solutions $\mathbf{x}_i(g)$. Note that the effect of the weights on the average is similar to the effect of a proportional selection (section 1.3.3) according to fitnesses f_i of the μ best solutions. In this way, $\mathbf{m}$ moves from generation to generation according to the optimizing path determined by the sequence of the sets of the best solutions $\mathbf{x}_i$ which have been selected. We have:

$$\mathbf{m}(g+1) = \sum_{i=1}^{\mu} f_i \mathbf{x}_i(g) \qquad [2.3]$$

2.3.1.3. *Adaptation of σ*

Step size $\sigma(g)$ is adapted so that successive vectors:

$$\delta(g+1) = \frac{\mathbf{m}(g+1) - \mathbf{m}(g)}{\sigma(g)}$$

according to g are decorrelated at best. Indeed, if the vectors $\delta(g)$ are strongly correlated (correlation coefficient close to 1), that means that $\sigma(g)$ is too small because each successive generation leads to progress in the search space almost in the same direction. Thus, $\sigma(g)$ should be increased, reducing the number of evaluations of the objective function for almost the same progression. Otherwise, if successive steps $\delta(g)$ are anticorrelated (correlation coefficient close to -1), this leads to variations of $\mathbf{m}(g)$ in almost opposite directions during successive generations, involving too slow progressions in the search space. It can be deduced from this situation that $\sigma(g)$ is too large.

To decide whether step size $\sigma(g)$ is too small or too large, the designers of the method used the notion of *evolution path* $\mathbf{p}_\sigma(g)$ that can be calculated as an average of $\delta(g)$ over a few generations. It is then compared to the average progression allowed by independent Gaussian random vectors drawn from the distribution $\delta(g)$. As the draws are independent, they are uncorrelated.

From equation [2.3], $\delta(g+1)$ can be rewritten as:

$$\delta(g+1) = \frac{\sum_{i=1}^{\mu} f_i \mathbf{x}_i(g) - \mathbf{m}(g)}{\sigma(g)}$$

By definition, the distribution of $\mathbf{x}_i(g)$ is $\mathcal{N}(\mathbf{m}(g), \sigma(g)^2 \mathbf{C}'(g))$. So, the distribution of $\sum_{i=1}^{\mu} f_i \mathbf{x}_i(g)$ is:

$$\sum_{i=1}^{\mu} f_i \mathbf{x}_i(g) \sim \mathcal{N}(\mathbf{m}(g) \sum_{i=1}^{\mu} f_i, \sigma(g)^2 \sum_{i=1}^{\mu} f_i^2 \mathbf{C}'(g))$$

Since $\sum_{i=1}^{\mu} f_i = 1$, and by defining $\mu_f = 1/\sum_{i=1}^{\mu} f_i^2$:

$$\sum_{i=1}^{\mu} f_i \mathbf{x}_i(g) \sim \mathcal{N}(\mathbf{m}(g), \sigma(g)^2 \mathbf{C}'(g)/\mu_f)$$

Thus,

$$\delta(g+1) \sim \mathcal{N}(0, \mathbf{C}'(g)/\mu_f)$$

$\delta(g+1)$ is a random vector drawn from the distribution $\mathcal{N}(0, \mathbf{C}'/\mu_f)$. In practice, a vector $\delta'(g+1)$ drawn from the distribution $\mathcal{N}(0, \mathbf{I}/\mu_f)$ is calculated as follows:

$$\delta'(g+1) = \mathbf{C}'(g)^{-1/2} \delta(g+1) = \mathbf{B}\mathbf{D}^{-1}\mathbf{B}^{\mathsf{T}} \delta(g+1)$$

where $\mathbf{B}$ and $\mathbf{D}$ are respectively the matrix of eigenvectors and the corresponding diagonal matrix of the square roots of the eigenvalues of $\mathbf{C}'(g)$. Thus, $\sqrt{\mu_f}\delta'(g+1)$ is drawn from the distribution $\mathcal{N}(0, \mathbf{I})$. The designers of the method propose to calculate recursively a weighted average of $\mathbf{p}_\sigma(g)$ and $\sqrt{\mu_f}\delta'(g+1)$ to obtain $\mathbf{p}_\sigma(g+1)$:

$$\mathbf{p}_\sigma(g+1) = (1 - c_\sigma)\mathbf{p}_\sigma(g) + \alpha\sqrt{\mu_f}\delta'(g+1)$$

$c_\sigma \in]0, 1[$ is a parameter of the method. Choosing c_σ close to 0 leads to a smooth but slow adaptation of $\mathbf{p}_\sigma$: the memory effect is important. α is calculated so that when step size $\sigma(g)$ is well-adapted, $\mathbf{p}_\sigma(g)$ and $\mathbf{p}_\sigma(g+1)$ have the same distribution $\mathcal{N}(0, \mathbf{I})$. Now, $\sqrt{\mu_f}\delta'(g+1)$ is also drawn from the distribution $\mathcal{N}(0, \mathbf{I})$. Therefore, $(1 - c_\sigma)^2 + \alpha^2 = 1$ so that the covariance matrix of $\mathbf{p}_\sigma(g+1)$ is $\mathbf{I}$. Consequently, $\alpha = \sqrt{1 - (1 - c_\sigma)^2}$. The expression of evolution path $\mathbf{p}_\sigma(g)$ with $g \geq 1$ is thereby obtained:

$$\begin{cases} \mathbf{p}_\sigma(g+1) = (1 - c_\sigma)\mathbf{p}_\sigma(g) + \sqrt{c_\sigma(2 - c_\sigma)\mu_f}\mathbf{B}\mathbf{D}^{-1}\mathbf{B}^\mathsf{T}\frac{\mathbf{m}(g+1) - \mathbf{m}(g)}{\sigma(g)} \\ \mathbf{p}_\sigma(1) = 0 \end{cases}$$

[2.4]

$||\mathbf{p}_\sigma(g+1)||$ is "compared" to $\mathsf{E}||\mathcal{N}(0, \mathbf{I})||$, which is the expectation of the norm of Gaussian random vectors drawn from the distribution $\mathcal{N}(0, \mathbf{I})$, to adapt the value of σ in such a way that it:

– decreases when $||\mathbf{p}_\sigma(g+1)||$ is less than $\mathsf{E}||\mathcal{N}(0, \mathbf{I})||$;

– increases when $||\mathbf{p}_\sigma(g+1)||$ is greater than $\mathsf{E}||\mathcal{N}(0, \mathbf{I})||$;

– remains constant when $||\mathbf{p}_\sigma(g+1)||$ is equal to $\mathsf{E}||\mathcal{N}(0, \mathbf{I})||$.

The following expression can efficiently perform this adaptation:

$$\sigma(g+1) = \sigma(g) \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||p_\sigma(g+1)||}{\mathsf{E}||\mathcal{N}(0, \mathbf{I})||} - 1\right)\right) \quad [2.5]$$

where d_σ is a damping factor, with a value of around 1. The value of $\sigma(0)$ is problem-dependent. c_σ, d_σ and $\sigma(0)$ are parameters of the method. A robust initialization strategy of these parameters is proposed on page 56.

2.3.1.4. *Adaptation of* $\mathbf{C}'$

The designers of the method have proposed an estimator $\mathbf{C}_\mu(g+1)$ for the covariance matrix $\mathbf{C}(g+1)$ from the μ best realizations $\mathbf{x}_i(g)$ obtained at generation g:

$$\mathbf{C}_\mu(g+1) = \sum_{i=1}^{\mu} f_i(\mathbf{x}_i - \mathbf{m}(g))(\mathbf{x}_i - \mathbf{m}(g))^\mathsf{T}$$

Note that this estimator uses the weighted average $\mathbf{m}(g)$ obtained in the previous generation instead of $\mathbf{m}(g+1)$. Moreover, the contribution of each term $(\mathbf{x}_i - \mathbf{m}(g))$ is weighted by $\sqrt{f_i}$. To see the relevance of this estimator intuitively on an example, we consider the case $\mu = 1$:

$$\mathbf{C}_1(g+1) = f_1(\mathbf{x}_1 - \mathbf{m}(g))(\mathbf{x}_1 - \mathbf{m}(g))^\mathsf{T}$$

The matrix $\mathbf{C}_1(g+1)$ has therefore only one non-zero eigenvalue, for an eigenvector collinear to $(\mathbf{x}_1 - \mathbf{m}(g))$. This means that the Gaussian distribution $\mathcal{N}(\mathbf{m}(g+1), \mathbf{C}_1(g+1))$ will generate realizations $\mathbf{x}_i(g+1)$ only on the line whose the direction vector is $(\mathbf{x}_1(g) - \mathbf{m}(g))$, passing through point $\mathbf{m}(g+1)$. Now, $\mathbf{x}_1(g)$ being the best solution obtained to the current generation, the heuristic choice of direction $(\mathbf{x}_1(g) - \mathbf{m}(g))$ to find a better solution $\mathbf{x}_1(g+1)$ is reasonable. However, *a priori*, this direction is not the one of the optimum. To ensure a good exploration of the search space, μ must be large enough, not less than n, so that the covariance matrix $\mathbf{C}_\mu(g+1)$ is positive definite.

Taking into account step size $\sigma(g)$, with $\mathbf{C}(g) = \sigma(g)^2\mathbf{C}'(g)$, the expression of $\mathbf{C}'_\mu(g+1)$ is:

$$\mathbf{C}'_\mu(g+1) = \sum_{i=1}^{\mu} f_i \frac{\mathbf{x}_i - \mathbf{m}(g)}{\sigma(g)} \left(\frac{\mathbf{x}_i - \mathbf{m}(g)}{\sigma(g)}\right)^\mathsf{T}$$

However, giving a large value to μ also increases the number of evaluations of the objective function needed to reach the optimum. To reduce the value of μ, while ensuring that the matrix $\mathbf{C}'(g+1)$ remains positive definite, it is possible to use the matrix $\mathbf{C}'(g)$ obtained in the previous generation.

2.3.1.4.1. Rank-μ update

The designers of the method propose that $\mathbf{C}'(g+1)$ is a weighted average of matrices $\mathbf{C}'(g)$ and $\mathbf{C}'_\mu(g+1)$, with respective weights $1-c_\mu$ and c_μ, where $c_\mu \in (0,1]$ is a parameter of the method:

$$\begin{cases} \mathbf{C}'(g+1) = (1-c_\mu)\mathbf{C}'(g) + c_\mu \mathbf{C}'_\mu(g+1) \\ \mathbf{C}'(1) = \mathbf{I} \end{cases} \qquad [2.6]$$

The matrix $\mathbf{C}'(g+1)$ is thereby defined by recurrence for $g \geq 1$. The identity matrix is chosen as the initial term because it is symmetric positive definite. By the recurrence relation, $\mathbf{C}'(g+1)$ is a weighted average of matrices $\mathbf{C}'_\mu(i)$ for $i \in \{1, ..., g+1\}$.

Thus, μ can be much smaller than n while keeping the matrices $\mathbf{C}'(g+1)$ positive definite. If c_μ is chosen close to 0, the matrix $\mathbf{C}'(g+1)$ depends strongly on the past and can accept small values of μ. But the evolution will be slow. If c_μ is chosen close to 1, the matrix $\mathbf{C}'(g+1)$ can evolve quickly, provided μ is large enough to ensure that the matrix $\mathbf{C}'$ remains positive definite, which ultimately increases the number of necessary evaluations of the objective function.

Expression [2.6] is suitable to update $\mathbf{C}'(g)$, but at the cost of an excessive number of generations with a value of μ which should be chosen large enough. To reduce the number of necessary evaluations of the objective function, an additional adaptation mechanism of $\mathbf{C}'(g)$ has been used.

2.3.1.4.2. Rank-one update

This adaptation mechanism of $\mathbf{C}'$ consists of generating in every generation a random vector $\mathbf{p}_c(g+1)$ according to the distribution $\mathcal{N}(0, \mathbf{C}')$. In section 2.3.1.3, we saw that $\delta(g+1) = (\mathbf{m}(g+1) - \mathbf{m}(g))/\sigma(g)$ has the distribution $\mathcal{N}(0, \mathbf{C}'/\mu_f)$. Similarly to $\mathbf{p}_\sigma(g+1)$, $\mathbf{p}_c(g+1)$ is expressed as an evolution path:

$$\begin{cases} \mathbf{p}_c(g+1) = (1-c_c)\mathbf{p}_c(g) + \sqrt{c_c(2-c_c)\mu_f}\, \frac{\mathbf{m}(g+1) - \mathbf{m}(g)}{\sigma(g)} \\ \mathbf{p}_c(1) = 0 \end{cases} \qquad [2.7]$$

$\mathbf{C}'(g+1)$, which must be of rank n, is expressed as a weighted average of $\mathbf{p}_c(g+1)\mathbf{p}_c(g+1)^{\mathsf{T}}$, which has rank 1, and of $\mathbf{C}'(g)$ of rank n:

$$\begin{cases} \mathbf{C}'(g+1) = (1-c_1)\mathbf{C}'(g) + c_1\mathbf{p}_c(g+1)\mathbf{p}_c(g+1)^{\mathsf{T}} \\ \mathbf{C}'(1) = \mathbf{I} \end{cases} \quad [2.8]$$

2.3.1.4.3. Update of $\mathbf{C}'$

The combination of rank-μ update (equation [2.6]) and rank-one update (equation [2.8]) expressions gives the complete expression of $\mathbf{C}'(g)$ updating for $g \geq 1$:

$$\begin{cases} \mathbf{C}'(g+1) = (1-c_1-c_\mu)\mathbf{C}'(g) + \\ \qquad\qquad + c_1\mathbf{p}_c(g+1)\mathbf{p}_c(g+1)^{\mathsf{T}} + c_\mu \sum_{i=1}^{\mu} f_i \mathbf{v}_i(g+1)\mathbf{v}_i(g+1)^{\mathsf{T}} \\ \mathbf{C}'(1) = \mathbf{I} \end{cases} \quad [2.9]$$

where $\mathbf{v}_i(g+1) = (\mathbf{x}_i - \mathbf{m}(g))/\sigma(g)$. c_c, c_1 and c_μ are parameters of the method. A robust initialization strategy of these parameters is provided on page 56.

2.3.2. *The CMA-ES algorithm*

Algorithm 1 implements the CMA-ES method as proposed by [HAN 06]. In every generation, λ independent (pseudo-) random solutions $\mathbf{x}_i$ are generated according to the distribution $\mathcal{N}(\mathbf{m}, \sigma^2\mathbf{C}')$ whose parameters were determined in the previous generation. The μ best solutions are sorted and returned by the function **Selection** (Algorithm 2) in the form of a matrix $\mathbf{X}$ of n rows and μ columns. Column i of $\mathbf{X}$ gives the solution $\mathbf{x}_i$. Column sorting is done so that if the objective value $F_i = F(\mathbf{x}_i)$ is better than $F_j = F(\mathbf{x}_j)$, then $i < j$.

From $\mathbf{X}$, the updates of parameters $\mathbf{m}$, σ and $\mathbf{C}'$ are assigned to the functions **UpdateM**, **UpdateSigma** and **UpdateC** (Algorithms 3, 4 and 5). These functions do not require special comments: their algorithms directly derive from analytical expressions given in the previous section.

Let $\mathbf{B}$ be the matrix whose columns i are eigenvectors $\mathbf{b}_i$ of $\mathbf{C}'$. Let $\mathbf{D}$ be the diagonal matrix such that d_{ii} is the square root of the eigenvalue of

$\mathbf{C}'$ corresponding to the eigenvector $\mathbf{b}_i$. Matrices $\mathbf{B}$ and $\mathbf{D}$ are required to compute $\mathbf{C}^{1/2}$ and to randomly draw solutions $\mathbf{X}$ according to the distribution $\mathcal{N}(\mathbf{m}, \sigma^2\mathbf{C}')$. The computation of $\mathbf{B}$ and $\mathbf{D}$ has $O(n^3)$ algorithmic complexity. It can be performed every $1/(10(c_1+c_\mu)n)$ generations as shown in algorithm 1 to reduce its average complexity to $O(n^2)$ [HAN 16].

```
Parameters:
$n$: dimension of the search space
$\mathbf{m}$: initial mean vector
$\sigma$: initial step-size

Result: $\mathbf{x}_1$: the best solution at the last generation
$\lambda, \mu, \mathbf{f}, \mu_f, c_\sigma, d_\sigma, c_c, c_1, c_\mu \leftarrow$ Initialization($n$)
$\mathbf{p}_c \leftarrow \mathbf{p}_\sigma \leftarrow 0$
$\mathbf{C}' \leftarrow \mathbf{B} \leftarrow \mathbf{D} \leftarrow \mathbf{I}$                    // $\mathbf{I}$: $n \times n$ identity matrix
$e_e \leftarrow c_e \leftarrow 0$                    // $c_e, e_e$: evaluation counters
repeat
    $\mathbf{X}, \mathbf{V} \leftarrow$Selection($\lambda, \mathbf{m}, \sigma, \mathbf{B}, \mathbf{D}$)
    $c_e \leftarrow c_e + 1$
    $\mathbf{m}, \delta \leftarrow$UpdateM($\mathbf{m}, \mu, \mathbf{X}, \mathbf{f}, \sigma$)
    $\sigma, \mathbf{p}_\sigma \leftarrow$UpdateSigma($\sigma, \mathbf{p}_\sigma, \mathbf{B}, \mathbf{D}, \delta, c_\sigma, d_\sigma, \mu_f$)
    $\mathbf{C}', \mathbf{p}_c \leftarrow$UpdateC($\mathbf{C}', \mathbf{p}_c, \mathbf{p}_\sigma, \mathbf{V}, \mathbf{f}, \delta, c_c, c_1, c_\mu, \mu, \mu_f$)
    if $c_e - e_e \geq \lfloor 1/(c_1 + c_\mu)/n/10 \rfloor$ then      // to achieve $O(n^2)$
        $e_e \leftarrow c_e$
        $\mathbf{B} \leftarrow$EigenVectors($\mathbf{C}'$)
        $\mathbf{D} \leftarrow$EigenValues($\mathbf{C}'$)$^{1/2}$
        // $\mathbf{D}$: diagonal matrix of the eigenvalue square roots
    end
until Stopping criterion satisfied
```

Algorithm 1: The CMA-ES algorithm

The setting of the algorithm parameters by the function **Initialization** *a priori* depends on the problem to solve. However, a default initialization, which has proven to be robust and effective, usable for many problems, was proposed by [HAN 06]. It is implemented by function **DefaultInitialization** (Algorithm 6). Hansen proposed another set of slightly different default initial values for the parameters in [HAN 16]. However, according to the problems, these initial values do not always lead to the best results. The values chosen for parameters λ, μ, $\mathbf{f} = (f_1, ..., f_\mu)$, c_σ, d_σ, c_c, c_1 and c_μ may be adapted to the problem to solve. Note that the proposed value for λ should be considered as a minimum. A larger value improves the robustness of the algorithm, especially for multimodal functions, at the cost, however, of a greater number of objective function evaluations.

```
for $i = 1$ to $\lambda$ do
    $\mathbf{y}_i \leftarrow \mathcal{N}(0, \mathbf{I})$    // $\mathbf{y}_i$ is a random vector with distribution $\mathcal{N}(0, \mathbf{I})$
    $\mathbf{v}_i \leftarrow \mathbf{BDy}_i$    // $\mathbf{v}_i$ has distribution $\mathcal{N}(0, \mathbf{C}')$ with $\mathbf{C}' = \mathbf{BD}^2\mathbf{B}^\mathsf{T}$
    $\mathbf{x}_i \leftarrow \mathbf{m} + \sigma \mathbf{v}_i$    // $\mathbf{x}_i$ has distribution $\mathcal{N}(\mathbf{m}, \sigma^2\mathbf{C}')$
    $F_i \leftarrow \text{Objective}(\mathbf{x}_i)$ // $F_i$ is the objective value associated with $\mathbf{x}_i$
end
$\mathbf{X}, \mathbf{V} \leftarrow \text{Sort}(\mathbf{X}, \mathbf{V}, \mathbf{F})$    // column sorting of $\mathbf{x}_i$ and $\mathbf{v}_i$ according to $F_i$
return $\mathbf{X}, \mathbf{V}$
```

Algorithm 2: Function Selection($\lambda, \mathbf{m}, \sigma, \mathbf{B}, \mathbf{D}$)

```
$\mathbf{m}' \leftarrow \mathbf{m}$
$\mathbf{m} \leftarrow \sum_{i=1}^{\mu} f_i \mathbf{x}_i$
$\delta \leftarrow (\mathbf{m} - \mathbf{m}')/\sigma$
return $\mathbf{m}, \delta$
```

Algorithm 3: Function UpdateM($\mathbf{m}, \mu, \mathbf{X}, \mathbf{f}, \sigma$)

```
$\mathbf{p}_\sigma \leftarrow (1 - c_\sigma)\mathbf{p}_\sigma + \sqrt{c_\sigma(2 - c_\sigma)\mu_f}\, \mathbf{B} \cdot \mathbf{D}^{-1} \cdot \mathbf{B}^\mathsf{T}\delta$
$\sigma \leftarrow \sigma \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||p_\sigma||}{\mathrm{E}||\mathcal{N}(0,\mathbf{I})||} - 1\right)\right)$
                                        // $\mathrm{E}||\mathcal{N}(0, \mathbf{I})|| \approx \sqrt{n}\left(1 - \frac{1}{4n} + \frac{1}{21n^2}\right)$
return $\sigma, \mathbf{p}_\sigma$
```

Algorithm 4: Function UpdateSigma($\sigma, \mathbf{p}_\sigma, \mathbf{B}, \mathbf{D}, \delta, c_\sigma, d_\sigma, \mu_f$)

The initial values of $\mathbf{m} = (m_1, ..., m_n)$ and σ depend on the problem. When the location of the optimum is approximately known, these initial values should be determined so that the optimum lies in the range defined by the intervals $[m_i - 2\sigma, m_i + 2\sigma]$ [HAN 06] for each coordinate $i \in \{1, ..., n\}$.

Simulation results obtained with CMA-ES on a set of various objective functions are presented in section 2.8).

2.4. A restart CMA Evolution Strategy

Hansen and Kern [HAN 04] noted that large populations significantly improve the probability of finding near-optimal solutions for some massively multimodal objective functions, such as the non-separable "rotated Rastrigin function" (sections 2.8.1.1 and 2.10), which counts 11^{100} local optima in search domain $[-100, 100]^{100}$. Table 2.1 shows the success rates and the median number of function evaluations (FEs) required to reach the near-optimal objective value of 0.01 according to four population sizes. 100 runs are performed for each result.

λ	FEs	Success rate
500	243,000	1%
1000	815,000	50%
2000	1,298,000	97%
4000	2,340,000	100%

Table 2.1. *Rotated Rastrigin function in search domain $[-100, 100]^{100}$: number of evaluations versus population size λ to reach an objective value less than 0.01 with the CMA-ES algorithm*

The default initialization of λ is 17 for this problem (Algorithm 6), although the optimal population size is near 2000 according to Table 2.1. These observations suggest to restart the CMA-ES algorithm by increasing the population size by a factor of two or three, until a stopping criterion is satisfied [HAN 04] [AUG 05]. The population size is initialized according to algorithm 6. This strategy is referred to as "IPOP-CMA-ES".

Obviously, the stopping criteria of the CMA-ES algorithm for a restart strategy should be more elaborate than just stopping the algorithm after a given number of function evaluations without improving the best solution found so far. To avoid wasting time in useless function evaluations before restarting, Auger and Hansen recommend some additional stopping criteria in [AUG 05] [HAN 09].

```
$\mathbf{p}_c \leftarrow (1 - c_c)\mathbf{p}_c$
if $||\mathbf{p}_\sigma|| < 1.5\sqrt{n}$ then
    $\mathbf{p}_c \leftarrow \mathbf{p}_c + \sqrt{c_c(2 - c_c)\mu_f}\ \delta$
end
$\mathbf{C}' \leftarrow (1 - c_1 - c_\mu)\mathbf{C}' + c_1\mathbf{p}_c\mathbf{p}_c^\mathsf{T} + c_\mu \sum_{i=1}^{\mu} f_i\mathbf{v}_i\mathbf{v}_i^\mathsf{T}$
return $\mathbf{C}', \mathbf{p}_c$
```

Algorithm 5: Function UpdateC($\mathbf{C}', \mathbf{p}_c, \mathbf{p}_\sigma, \mathbf{V}, \mathbf{f}, \delta, c_c, c_1, c_\mu, \mu, \mu_f$)

```
$\lambda \leftarrow 4 + \lfloor 3\ln n \rfloor$          // $\lfloor x \rfloor$ is the lower integer part of $x$
$\mu \leftarrow \lfloor \lambda/2 \rfloor$
for $i = 1$ to $\mu$ do
    $f_i \leftarrow \frac{\ln(\mu+1) - \ln i}{\sum_{j=1}^{\mu} \ln(\mu+1) - \ln j}$          // $\mathbf{f} = (f_1, ..., f_i, ..., f_\mu)$
end
$\mu_f \leftarrow 1/\sum_{i=1}^{\mu} f_i^2$
$c_\sigma \leftarrow \frac{\mu_f + 2}{n + \mu_f + 3}$
$d_\sigma \leftarrow 1 + 2\max\left(0, \sqrt{\frac{\mu_f - 1}{n+1}} - 1\right) + c_\sigma$
$c_c \leftarrow 4/(n + 4)$
$c_{\text{cov}} \leftarrow \frac{2}{\mu_f(n + \sqrt{2})^2} + \left(1 - \frac{1}{\mu_f}\right)\min\left(1, \frac{2\mu_f - 1}{(n+2)^2 + \mu_f}\right)$
$c_1 \leftarrow c_{\text{cov}}/\mu_f$
$c_\mu \leftarrow c_{\text{cov}} - c_1$
return $\lambda, \mu, \mathbf{f}, \mu_f, c_\sigma, d_\sigma, c_c, c_1, c_\mu$
```

Algorithm 6: Function DefaultInitialization(n)

Simulation results obtained with IPOP-CMA-ES on a set of various objective functions are presented in section 2.8.

2.5. Differential Evolution (DE)

The canonical "Differential Evolution" algorithm was presented in 1997 by R. Storn and K. Price [STO 97] to solve continuous optimization problems. The authors proposed an efficient, simple and robust method with few free parameters (Algorithm 7). A population is a set of N *target vectors* $\mathbf{x}_i$, $i \in \{1, ..., N\}$ defined in $\mathbb{R}^d$. During a generation, a *mutant* (or *donor*) vector $\mathbf{v}_i$ is generated by a mutation operator for each target vector $\mathbf{x}_i$. $\mathbf{x}_i$ is then recombined with $\mathbf{v}_i$ by a crossover operator to yield *trial vector* $\mathbf{u}_i$. Next, during the selection step (Algorithm 10), a new population of target vectors is built from the current population of target vectors and the set of trial vectors.

```
Parameters:
$N$: number of target vectors in the population
$\mathbf{lb}$: vector of lower bounds in the search space
$\mathbf{ub}$: vector of upper bounds in the search space
$F$: scaling factor for the mutation
CR: crossover rate

Result: $\mathbf{x}_b$: the best target vector in the population

Variables:
$\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$: population of target vectors $\mathbf{x}_i$
$\mathbf{v}$: mutant vectors generated by the mutation operator from a subset of
target vectors
$\mathbf{U} = \{\mathbf{u}_1, ..., \mathbf{u}_N\}$: set of the trial vectors generated by the crossover
operator
$\mathbf{X} \leftarrow \text{Initialize}(N, \mathbf{lb}, \mathbf{ub})$
repeat
    for $i = 1$ to $N$ do
        $\mathbf{v} \leftarrow \text{Mutation}(\mathbf{X}, i, F)$
        $\mathbf{u}_i \leftarrow \text{Crossover}(\mathbf{x}_i, \mathbf{v}, \text{CR})$
    end
    $\mathbf{X} \leftarrow \text{Selection}(\mathbf{X}, \mathbf{U})$
until Stopping criterion satisfied
```

Algorithm 7: The canonical Differential Evolution algorithm

Several Differential Evolution variants have been proposed, whose efficiencies are problem-dependent. They can be identified according to the following notation: "DE/m/p/c" where m indicates the mutation variant, p is the number of difference vectors involved to obtain a mutant vector and c refers to the crossover variant. For instance, "DE/rand/1/bin" refers to a differential algorithm implementing the "rand" mutation operator with 1 difference vector and the "bin" crossover operator.

2.5.1. *Initializing the population*

If each variable x_j of the optimization problem is defined in an interval $[l_j, u_j]$, the N target vectors of the population are often randomly initialized according to a uniform distribution in the domain $\prod_{j=1}^{d}[l_j, u_j]$. Such initialization in a bounded domain can also be used when the values of x_j are unbounded, but the optimum should be located inside this domain. Otherwise, the risk that the algorithm will be trapped by a local optimum increases for multimodal functions. Using a Gaussian distribution is not recommended [PRI 05].

2.5.2. *The mutation operator*

The originality and the efficiency of Differential Evolution essentially lie in its mutation operator. For each target vector $\mathbf{x}_i$ of the population, the first proposed version of this operator randomly chooses three other target vectors $\mathbf{x}_{R1}$, $\mathbf{x}_{R2}$ and $\mathbf{x}_{R3}$, such that $R1$, $R2$ and $R3$ are mutually different indexes drawn from the set $\{1, ..., N\} \setminus \{i\}$ [PRI 05]. A *difference vector* is computed as $\mathbf{x}_{R2} - \mathbf{x}_{R3}$. It is then weighted by the mutation *scaling factor* F and added to *base vector* $\mathbf{x}_{R1}$ to yield a *mutant vector* $\mathbf{v}$:

$$\mathbf{v} = \mathbf{x}_{R1} + F(\mathbf{x}_{R2} - \mathbf{x}_{R3}) \qquad [2.10]$$

Figure 2.6 shows the distributions of difference vectors as scatter plots for two given populations of target vectors. A complete set of difference vectors for a given population presents a central symmetry in the origin. The shape of a difference vector set is roughly similar to the shape of the target vector population that has generated it. In this way, if the population is mainly located in a promising area of the search space, the diversity generated by the difference vectors adapts itself to mainly explore this area and its near neighborhood. This behavior is different, for example, from a Gaussian mutation where a covariance matrix must be given to define the exploration neighborhood (section 2.2.2.6).

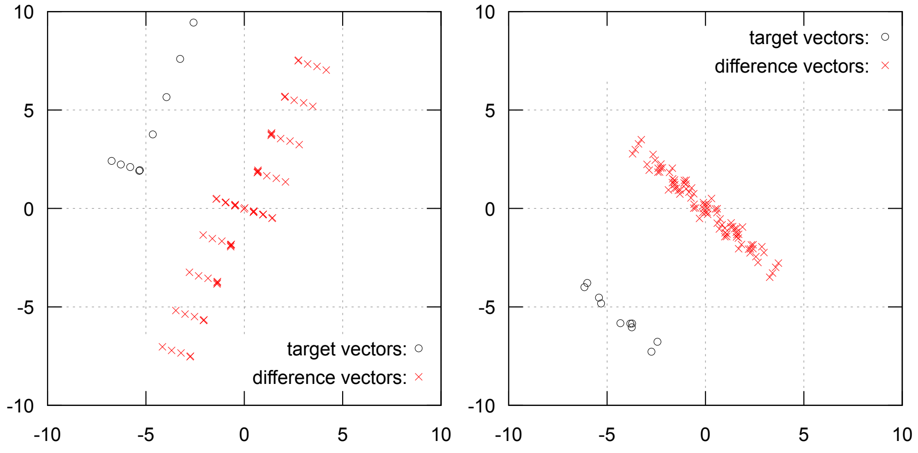


Figure 2.6. *Distributions of difference vectors generated by the DE/rand/1 mutation operator from two populations of target vectors*

Price *et al*. empirically noted that $F > 1$ leads to less efficient progression towards the optimum of objective functions. Furthermore, theoretical considerations about the preservation of the diversity in the population suggest that:

$$F > \sqrt{\frac{2 - p_c}{2N}}$$

where p_c is the crossover probability, taken equal to parameter CR [ZAH 02]. So, $F > 0.1341$ when $N = 50$ and $p_c = 0.2$. However, to avoid frequent premature convergences, empirical studies on three simple benchmark problems have shown that F should be equal to or greater than 0.3 for the same parameter values [ZAH 02].

F can be randomized to increase diversity among mutant vectors. Such a choice can be useful or harmful according to the properties of the objective functions [PRI 05]. Randomizing F (as well as CR) can be also used as a way to explore the parameter space in self-adaptative strategies of Differential Evolution [DAS 16].

The mutation defined by equation [2.10] is the DE/rand/1 variant, where "rand" means that the base vectors are chosen at random in the population. The most common variants [DAS 16] are listed below:

DE/rand/p	$\mathbf{v} = \mathbf{x}_{R(1)} + F\sum_{j=1}^{p}(\mathbf{x}_{R(2j)} - \mathbf{x}_{R(2j+1)})$
DE/best/p	$\mathbf{v} = \mathbf{x}_b + F\sum_{j=1}^{p}(\mathbf{x}_{R(2j-1)} - \mathbf{x}_{R(2j)})$
DE/current-to-best/p	$\mathbf{v} = \mathbf{x}_i + \alpha(\mathbf{x}_b - \mathbf{x}_i)$ $+F\sum_{j=1}^{p}(\mathbf{x}_{R(2j-1)} - \mathbf{x}_{R(2j)}),\ \alpha \in [0,1]$

$R(n)$ are random mutually different indices of target vectors in the population. $\mathbf{x}_b$ is the target vector with the best fitness. For the DE/best/p variant, the base vector is $\mathbf{x}_b$. This variant has the advantage of increasing the convergence speed, but it increases also the premature convergence risk. The DE/current-to-best/p variant reduces the high selection pressure of DE/best/p according to α by performing a convex combination of $\mathbf{x}_i$ and current target vector $\mathbf{x}_b$ to obtain the base vectors. This operation is similar to a BLX-0 crossover (section 2.2.1.2.2). Extra parameter α can be removed by replacing it with F in the expression of $\mathbf{v}$.

The sum of several difference vectors can be used to increase diversity among the mutant vectors when the population of target vectors is small. The number of difference vectors p is usually equal to 1 or, less frequently, 2. Figure 2.7 shows scatter plots of the sums of two difference vectors generated by the same populations as in Figure 2.6.

2.5.3. *The crossover operator*

Differential Evolution uses a crossover operator that combines each target vector $\mathbf{x}_i$ of the population with its associated mutant vector $\mathbf{v}$ with probabilities depending on CR to produce a trial vector $\mathbf{u}_i$. The following two variants are widely used: the *binomial crossover* and the *exponential crossover*. These variants ensure that at least one component of the mutant vector is copied in the trial vector. This disposition contributes to more efficiently explore the search space during an evolution.

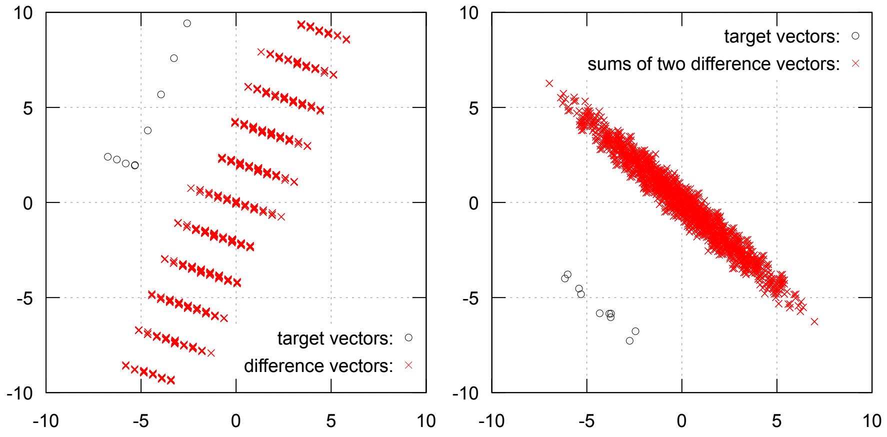


Figure 2.7. *Distributions of sums of two randomly chosen difference vectors generated by the DE/rand/2 mutation operator from two populations of target vectors*

2.5.3.1. *Binomial crossover*

The binomial crossover operator is described by Algorithm 8. Notation "$x \leftarrow \mathcal{U}[1, d+1)$" means that x is a random number that follows the distribution $\mathcal{U}[1, d+1)$.

```
$d \leftarrow \text{dimension}(\mathbf{x}_i)$
$K \leftarrow \lfloor \mathcal{U}[1, d+1) \rfloor$                    // $K \in \{1, ..., d\}$
**for** $j = 1$ **to** $d$ **do**
    **if** $\mathcal{U}(0,1) < \text{CR}$ **or** $j = K$ **then**
        $u_{ij} \leftarrow v_j$
    **end**
    **else**
        $u_{ij} \leftarrow x_{ij}$
    **end**
**end**
**return** $\mathbf{u}_i$                    // $\mathbf{u}_i = \{u_{i1}, ..., u_{id}\}$
```

Algorithm 8: Function BinCrossover($\mathbf{x}_i$, $\mathbf{v}$, CR)

This operator is similar to uniform crossover (section 2.2.1.1). Basically, each component of trial vector u_{ij} is v_j with probability CR and u_{ij} otherwise. In this case, the number of components u_{ij} coming from $\mathbf{v}$ follows a binomial distribution, which has given its name to the operator. However, with a binomial distribution, there is a non-zero probability that $\mathbf{u}_i = \mathbf{x}_i$. To eliminate this possibility, an index K is randomly chosen in $\{1, ..., d\}$ to assign v_K to u_{iK}. Then, the probability $p(x = m; d)$ that there are exactly m components coming from the mutant vector to build a trial vector is:

$$p(x = m; d) = C_{m-1}^{d-1} \mathrm{CR}^{m-1} (1 - \mathrm{CR})^{d-m}$$

where C_k^n is the number of combinations of n elements taken k at a time. Because of the choice of a vector component given by a random index K, CR is no longer equal to probability p_c for copying a component v_j in the trial vector. For example, when $\mathrm{CR} = 0$, then $p(x = 1; d) = 1$ and $p_c = 1/d$.

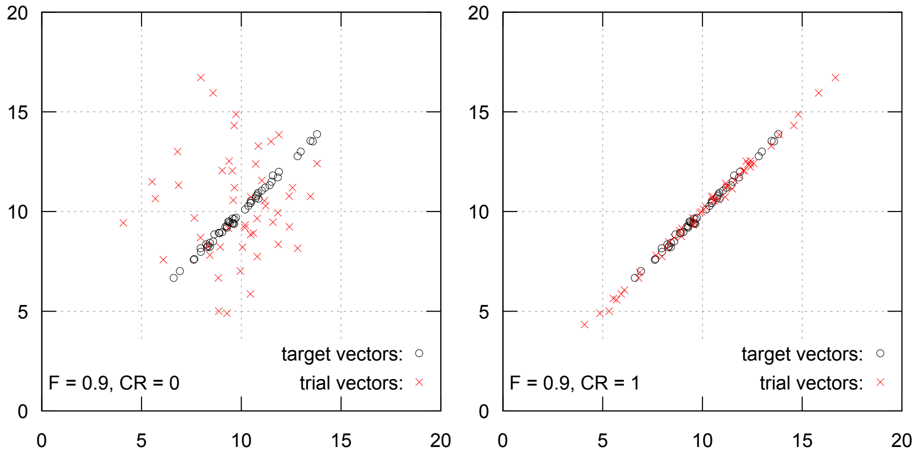


Figure 2.8. *Population of target vectors and set of trial vectors after the application of DE/rand/1/bin mutation and crossover operators, with F = 0.9, CR = 0 (left) and CR = 1 (right)*

Figure 2.8 shows scatter plots of populations of trial vectors and target vectors just after the application of the crossover operator in a 2D search space. The target vector population is the same for both figures. Such distribution of target vectors can be met when target vectors are located in near optimal values of non-separable, ill-conditioned quadratic objective functions, where contour lines are ellipses.

On the left-hand side, the trial vectors are computed with CR $= 0$. So, every trial vector is obtained with one component of a target vector and one component of a mutant vector. The figure shows that trial vectors are uncorrelated although target vectors are correlated. This means that, if target vector objective values are near the optimum of a non-separable, ill-conditioned function, most of the trial vectors will be discarded by the selection operator. In this case, the exploration quality brought by trial vectors is bad. More generally, choosing low values for CR is often better suited for separable objective functions [PRI 05].

The right-hand side shows the trial vectors when CR = 1. In this case, the trial vectors are also the mutant vectors. They are well correlated with target vectors. If these ones are distributed in a promising area of the objective function, then trial vectors explore efficiently this area. More generally, choosing CR values close to 1 is often better suited for non-separable objective functions.

2.5.3.2. *Exponential crossover*

The exponential crossover, described by Algorithm 9, is similar to the two-point crossover used by the genetic algorithms. A set of consecutive components of a mutant vector $\mathbf{v}$ is copied in trial vector $\mathbf{u}_i$ from a randomly chosen index K. The copy stops as soon as a random number drawn from the interval [0, 1] for each component is greater than CR. The other components of $\mathbf{u}_i$ are obtained from the target vector $\mathbf{x}_i$.

```
$d \leftarrow \text{dimension}(\mathbf{x}_i)$
$K \leftarrow \lfloor \mathcal{U}[1, d+1) \rfloor$
$j \leftarrow K$
**do**
    $u_{ij} \leftarrow v_j$
    $j \leftarrow j \bmod d + 1$
**while** $\mathcal{U}[0,1] < \text{CR}$ **and** $j \neq K$

**while** $j \neq K$ **do**
    $u_{ij} \leftarrow x_{ij}$
    $j \leftarrow j \bmod d + 1$
**end**
**return** $\mathbf{u}_i$
```

Algorithm 9: Function ExpCrossover($\mathbf{x}_i$, $\mathbf{v}$, CR)

The probability $p(x = m; d)$ that there are exactly m components coming from the mutant vector to build a trial vector with exponential crossover is:

$$p(x = m) = (1 - \mathrm{CR})\mathrm{CR}^{m-1}$$

Exponential crossover is less frequently used than binomial crossover [DAS 16]. Indeed, in most cases for continuous optimization problems, there is *a priori* no reason to preserve blocks of consecutive components of a vector.

2.5.4. *The selection operator*

The selection operator, described in Algorithm 10, builds a new population of target vectors from the current target vectors and the set of trial vectors generated by the crossover and mutation operators.

```
for $i = 1$ to $N$ do
    if $f(\mathbf{u}_i) \leq f(\mathbf{x}_i)$ then        // “$\leq$” replaced by “$\geq$” for maximizations
        $\mathbf{x}_i \leftarrow \mathbf{u}_i$
    end
end
return $\mathbf{X}$
```

Algorithm 10: Function Selection($\mathbf{X}$, $\mathbf{U}$)

For each index i, if the objective value associated with $\mathbf{u}_i$ is better than the objective value of target vector $\mathbf{x}_i$, $\mathbf{u}_i$ replaces $\mathbf{x}_i$ in the population. In the other case, $\mathbf{u}_i$ is discarded. “Better than” stands for “greater than” in a maximization context and “less than” in a minimization context. This “one to one selection” is a kind of deterministic binary tournament where a parent (target vector) and its unique child (trial vector) compete to be present in the population at the next generation. Such a selection is elitist because it keeps the best solution so far in the population along an evolution. It is also not subject to the genetic drift phenomenon, which can lead to loss of good solutions (section 1.3.2).

Simulation results obtained with the canonical Differential Evolution algorithm on a set of various objective functions are presented in section 2.8.

2.6. Success-History based Adaptive Differential Evolution (SHADE)

The *Success-History based Adaptive Differential Evolution* Algorithm [TAN 13b] is a Differential Evolution variant, which is among the top-ranked evolutionary algorithms in competitions on real parameter optimization [TAN 13a].

Numerous authors have noted that values of scaling factor F and crossover rate CR are critical according to the objective function [DAS 16]. Thus, the main improvement of SHADE from the canonical DE algorithm lies in self-adaptation of parameters F and CR. Moreover, SHADE uses the non-standard *current-to-pbest mutation* proposed for the JADE algorithm [ZHA 09] to increase diversity among mutant vectors.

2.6.1. *The algorithm*

In addition to the canonical DE, the SHADE algorithm uses two histories $\mathbf{M}_F$ and $\mathbf{M}_{\mathrm{CR}}$ that contain mean values of F and CR computed in past generations. They are used to generate random individual scaling factor F_i and crossover rate CR_i for each current target vector $\mathbf{x}_i$ with function "GetFromHistory" (Algorithm 11). When all trial vectors are computed during a generation, the "UpdateHistory" function updates the oldest entry in the histories if at least one trial vector $\mathbf{u}_i$ is better than $\mathbf{x}_i$: these histories are updated on success.

SHADE can also use an optional archive $\mathbf{A}$ containing target vectors of past generations to increase the diversity of the difference vectors required by the mutation. Target vectors $\mathbf{x}_i$ are moved to $\mathbf{A}$ from population $\mathbf{X}$ when the corresponding trial vector $\mathbf{u}_i$ is better than $\mathbf{x}_i$. This task is implemented by function "UpdateArchive".

```
**Parameters**:
$N$: number of target vectors in the population
$S_M$: number of values in histories $\mathbf{M}_{\mathrm{CR}}$ and $\mathbf{M}_F$
$S_A$: maximal size of external archive $\mathbf{A}$
$\mathbf{lb}$: vector of lower bounds in the search space
$\mathbf{ub}$: vector of upper bounds in the search space

**Result**: $\mathbf{x}_b$: the best target vector in the population

**Variables**:
$\mathbf{M}_F$: success history of scaling factors
$\mathbf{M}_{\mathrm{CR}}$: success history of crossover rates
$\mathbf{A}$: archive of target vectors in past generations
$\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$: population of target vectors $\mathbf{x}_i$
$\mathbf{v}$: mutant vector generated by the mutation operator from a subset of
target vectors
$\mathbf{U} = \{\mathbf{u}_1, ..., \mathbf{u}_N\}$: set of the trial vectors generated by the crossover
operator

$\mathbf{X}, \mathbf{A}, \mathbf{M}_{\mathrm{CR}}, \mathbf{M}_F \leftarrow \mathrm{Initialize}(N, \mathbf{lb}, \mathbf{ub}, S_M)$
repeat
    for $i = 1$ to $N$ do
        $F_i, \mathrm{CR}_i \leftarrow \mathrm{GetFromHistory}(\mathbf{M}_F, \mathbf{M}_{\mathrm{CR}})$
                                                    // $\mathbf{F} = \{F_1, ..., F_N\}$
        $\mathbf{v} \leftarrow \mathrm{Mutation}(\mathbf{X}, i, F_i, \mathbf{A})$              // $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$
        $\mathbf{u}_i \leftarrow \mathrm{Crossover}(\mathbf{x}_i, \mathbf{v}, \mathrm{CR}_i)$          // $\mathbf{CR} = \{\mathrm{CR}_1, ..., \mathrm{CR}_N\}$
    end
    $\mathbf{A} \leftarrow \mathrm{UpdateArchive}(\mathbf{A}, S_A, \mathbf{X}, \mathbf{U})$
    $\mathbf{M}_{\mathrm{CR}}, \mathbf{M}_F \leftarrow \mathrm{UpdateHistory}(\mathbf{M}_{\mathrm{CR}}, \mathbf{M}_F, \mathbf{X}, \mathbf{U}, \mathbf{CR}, \mathbf{F})$
    $\mathbf{X} \leftarrow \mathrm{Selection}(\mathbf{X}, \mathbf{U})$
until *Stopping criterion satisfied*
```

Algorithm 11: The SHADE algorithm

The target vectors are initialized in the same way as the canonical DE (section 2.5.1). In addition, the “Initialize” function (Algorithm 12) sets all values of F_k and CR_k to 0.5 in histories $\mathbf{M}_F$ and $\mathbf{M}_{\mathrm{CR}}$. Archive $\mathbf{A}$ is empty at the beginning. The crossover operator chosen for SHADE is the binomial variant (Algorithm 8). The selection operator is identical to that of the canonical DE (Algorithm 10). The mutation operator is specific and is described below as well as the functions for managing the success histories and the archive.

```
$\mathbf{X} \leftarrow \mathrm{RandomInit}(N, \mathbf{lb}, \mathbf{ub})$
$\mathbf{M}_{\mathrm{CR}} \leftarrow \mathrm{Init}(S_M, 0.5)$   // values in $\mathbf{M}_{\mathrm{CR}}$ initialized to 0.5
$\mathbf{M}_F \leftarrow \mathrm{Init}(S_M, 0.5)$     // values in $\mathbf{M}_F$ initialized to 0.5
$\mathbf{A} \leftarrow \emptyset$  // external archive used by the mutation operator
return $\mathbf{X}, \mathbf{A}, \mathbf{M}_{\mathrm{CR}}, \mathbf{M}_F$
```

Algorithm 12: Function Initialize(N, **lb**, **ub**, S_M)

2.6.2. *Current-to-pbest/1 mutation*

The *current-to-pbest/1* mutation has been proposed for the JADE algorithm [ZHA 09]. It can be considered as a generalization of the *current-to-best/1* mutation (section 2.5.2). The expression of the trial vector $\mathbf{v}$ obtained with this mutation is given as:

$$\mathbf{v} = \mathbf{x}_i + F(\mathbf{b}_p - \mathbf{x}_i) + F(\mathbf{x}_{R1} - \mathbf{y})$$

Instead of using the best target vector $\mathbf{x}_\mathbf{b}$, the operator chooses randomly $\mathbf{b}_p$ among the p best target vectors of the population, where p is a uniform random integer drawn from the set $\{2, ..., \lfloor N/5 \rfloor\}$, assuming that $N \geq 10$. So, at the difference of the *current-to-best/1* variant, the greediness of *current-to-pbest/1* mutation is adjustable: it decreases as p increases.

$\mathbf{y}$ is randomly drawn from $\mathbf{X} \cup \mathbf{A}$, where $\mathbf{A}$ is an archive of target vectors removed from the population by the selection operator in recent past generations. Even though the vectors in $\mathbf{A}$ are not the best ones, they bring useful information to progress towards the optimum. Also, it increases the diversity among difference vectors.

If the archive is full, i.e. $|\mathbf{A}| = S_A$, a new target vector added in $\mathbf{A}$ replaces an old one taken at random according to Algorithm 13.

```
$N \leftarrow |\mathbf{X}|$
for $i = 1$ to $N$ do
    if $f(\mathbf{u}_i) < f(\mathbf{x}_i)$ then    // maximization: "<" replaced by ">"
        if $|\mathbf{A}| \geq S_A$ then
            $r \leftarrow \lfloor \mathcal{U}[1, S_A + 1) \rfloor$    // $r \in \{1, ..., S_A\}$
        end
        else
            $r \leftarrow |\mathbf{A}| + 1$
        end
        $\mathbf{a}_r \leftarrow \mathbf{x}_i$    // $\mathbf{A} = \{\mathbf{a}_1, ..., \mathbf{a}_N\}$
    end
end
return $\mathbf{A}$
```

Algorithm 13: Function UpdateArchive($\mathbf{A}$, S_A, $\mathbf{X}$, $\mathbf{U}$)

2.6.3. *The success history*

2.6.3.1. *GetFromHistory function*

This function attributes values to scalar factor F_i and crossover rate CR_i associated with target vector $\mathbf{x}_i$. It is described by Algorithm 14. First, the function chooses at random the r-th recorded couple $(\mathbf{M}_{F,r}, \mathbf{M}_{\mathrm{CR},r})$ in histories $\mathbf{M}_F$ and $\mathbf{M}_{\mathrm{CR}}$. F_i is a random number that follows a truncated Cauchy distribution $\mathcal{C}_{[0,1]}(m, s)$ between 0 and 1 with median $m = \mathbf{M}_{F,r}$ and scale $s = 0.1$. $\mathcal{C}_{[0,1]}(\mathbf{M}_{F,r}, 0.1)$ values are generated while they are zero or negative, to obtain a value for F_i. Otherwise, $F_i = 1$ if $\mathcal{C}_{[0,1]}(\mathbf{M}_{F,r}, 0.1) > 1$. A Cauchy distribution for F_i is preferred to a Gaussian distribution to obtain more realizations that are far from the median with a narrow peak located at the median, as shown in Figure 2.9. This choice contributes to reduce the premature convergence risk [ZHA 09].

$\mathbf{M}_{\mathrm{CR},r}$ can have a special "terminal value" noted $\perp$. In this case: $\mathrm{CR}_i = 0$. Otherwise, CR_i is a random number that follows a truncated Gaussian distribution between 0 and 1 with mean equal to $\mathbf{M}_{\mathrm{CR},r}$ and standard deviation set to 0.1. CR_i is set to 0 if $\mathcal{N}(\mathbf{M}_{\mathrm{CR},r}, 0.1) < 0$ or CR_i is set to 1 if $\mathcal{N}(\mathbf{M}_{\mathrm{CR},r}, 0.1) > 1$.

```
$H \leftarrow |\mathbf{M}_F|$
$r \leftarrow \lfloor \mathcal{U}[1, H+1) \rfloor$                                  // $r \in \{1, ..., H\}$
repeat
    $F \leftarrow \mathcal{C}_{[0,1]}(\mathbf{M}_{F,r}, 0.1)$ // $\mathcal{C}_{[0,1]}(m, s)$: Cauchy random number
until $F > 0$
if $F > 1$ then
    $F \leftarrow 1$
end

if $\mathbf{M}_{\mathrm{CR},r} = \perp$ then
    $\mathrm{CR} \leftarrow 0$
end
else
    $\mathrm{CR} \leftarrow \mathcal{N}(\mathbf{M}_{\mathrm{CR},r}, 0.1)$ // $\mathcal{N}(m, \sigma)$: Gaussian random number
end
if $\mathrm{CR} < 0$ then
    $\mathrm{CR} \leftarrow 0$
end
if $\mathrm{CR} > 1$ then
    $\mathrm{CR} \leftarrow 1$
end
return $F, \mathrm{CR}$
```

Algorithm 14: Function GetFromHistory$(\mathbf{M}_F, \mathbf{M}_{\mathrm{CR}})$

2.6.3.2. *Success history updating*

The success histories $\mathbf{M}_F$ and $\mathbf{M}_{\mathrm{CR}}$ are updated before applying the selection operator when all trial vectors $\mathbf{u}_i$ at the current generation are computed. Let $\mathbf{M}_{F,k}$ and $\mathbf{M}_{\mathrm{CR},k}$ be the oldest values in the histories. These entries will be updated only if mutation and crossover operators have generated vectors $\mathbf{u}_i$ better than $\mathbf{x}_i$.

If all CR_i are zero or if $\mathbf{M}_{\mathrm{CR},k} = \perp$, then $\mathbf{M}_{\mathrm{CR},k}$ is updated with $\perp$. In this way, CR_i values are locked to zero for the remaining generations. This is useful to improve convergence for multimodal problems [TAN 14].

For other cases, values of $\mathbf{M}_{F,k}$ and $\mathbf{M}_{\mathrm{CR},k}$ are replaced respectively by the weighted Lehmer means of F_i and CR_i for all $i \in \{1, ..., N\}$ (Algorithm 15).

The Lehmer mean is preferred to the arithmetic mean to increase the progress rate [ZHA 09]. Weight δ_i for Lehmer means is $\|f(\mathbf{u}_i) - f(\mathbf{x}_i)\|$ if $f(\mathbf{u}_i)$ is better than $f(\mathbf{x}_i)$. Otherwise, $\delta_i = 0$.

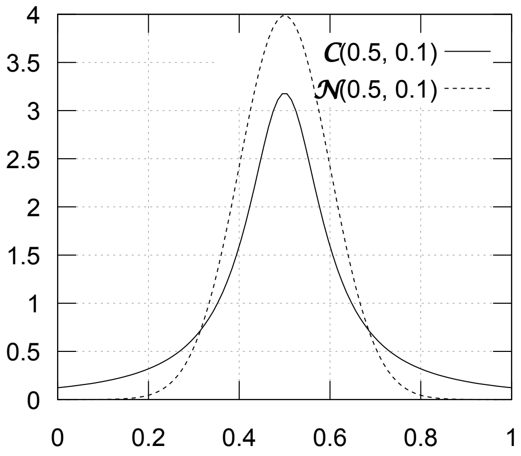


Figure 2.9. *Probability density functions for the Cauchy distribution* $\mathcal{C}(0.5, 0.1)$ *and for the Gaussian distribution* $\mathcal{N}(0.5, 0.1)$ *in the interval* $[0, 1]$

Simulation results obtained with SHADE on a set of various objective functions are presented in section 2.8.

2.7. Particle Swarm Optimization

Particle Swarm Optimizers (PSOs) take their inspiration from the collective behavior of animals such as schools of fish or flocks of birds. More generally, the designers [KEN 95] of the first PSO note that such collective behaviors are observed among groups of intelligent individuals on many aspects, even for example in the case of social relations between humans. A PSO comes from an extremely simplified model of social behavior.

Each individual, called a *particle*, remembers its best experience and takes advantage of the best experiences of its neighbors to explore its environment and improve its profit. The experience is modeled by a position $\mathbf{x}_i(t) = (x_{i1}(t), ..., x_{id}(t))$ at time t for particle i in a search space, which is typically a domain of $\mathbb{R}^d$. A fitness value $f(\mathbf{x}_i(t)) \in \mathbb{R}$ is associated with this position. Each particle moves in this search space with a *velocity* $\mathbf{v}_i(t) = (v_{i1}(t), ..., v_{id}(t))$. At each instant of time, particles adapt their velocities by combining two objectives: coming back to their best memorized positions $\mathbf{b}_i(t) = (b_{i1}(t), ..., b_{id}(t))$ and going towards the best positions $\mathbf{n}_i(t) = (n_{i1}(t), ..., n_{id}(t))$ memorized among their neighbors.

```
persistent $k = 1$ // $k$ persistent between calls to UpdateHistory
          // $k$ initialized to 1 before the first function call
$N \leftarrow |\mathbf{X}|$
for $i = 1$ to $N$ do
    if $f(\mathbf{u}_i) < f(\mathbf{x}_i)$ then  // maximizations: "$<$" replaced by "$>$"
        $\delta_i \leftarrow |f(\mathbf{u}_i) - f(\mathbf{x}_i)|$
    end
    else
        $\delta_i \leftarrow 0$
    end
end
if $\exists i \in \{1, ..., N\}, f(\mathbf{u}_i) < f(\mathbf{x}_i)$ then        // "$>$" for maximizations
    if $\forall i \in \{1, ..., N\}, \mathrm{CR}_i = 0$ then
        $\mathbf{M}_{\mathrm{CR},k} \leftarrow \perp$
    end
    else if $\mathbf{M}_{\mathrm{CR},k} \neq \perp$ then
        $\mathbf{M}_{\mathrm{CR},k} \leftarrow (\sum_{i=1}^{N} \delta_i \mathrm{CR}_i^2) / \sum_{i=1}^{N} \delta_i \mathrm{CR}_i$
                                              // weighted Lehmer mean of $\mathrm{CR}_i$
    end
    $\mathbf{M}_{F,k} \leftarrow (\sum_{i=1}^{N} \delta_i F_i^2) / (\sum_{i=1}^{N} \delta_i F_i)$
                                                // weighted Lehmer mean of $F_i$
    $k \leftarrow (k \bmod |\mathbf{M}_{\mathrm{CR}}|) + 1$                 // note: $|\mathbf{M}_{\mathrm{CR}}| = |\mathbf{M}_F|$
end
return $\mathbf{M}_{\mathrm{CR}}$, $\mathbf{M}_F$
```

Algorithm 15: Function UpdateHistory($\mathbf{M}_{\mathrm{CR}}$, $\mathbf{M}_F$, $\mathbf{X}$, $\mathbf{U}$, $\mathbf{CR}$, $\mathbf{F}$)

The flowchart in Figure 2.10 shows the basic building blocks of PSO.

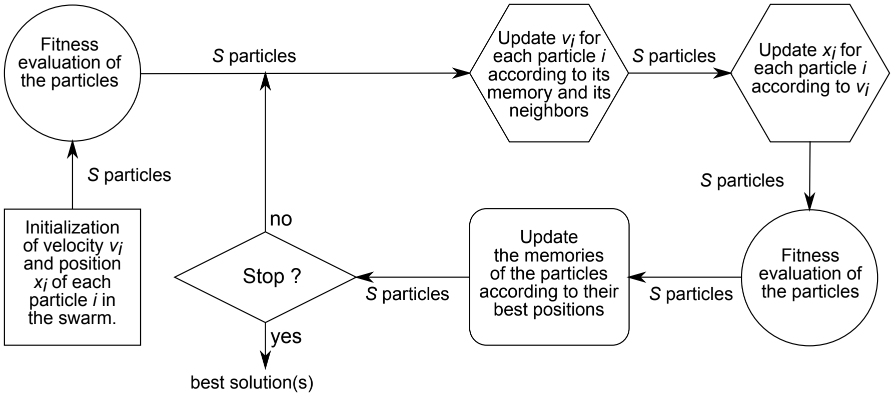

Fitness evaluation of the particles
S particles
Update v_i for each particle i according to its memory and its neighbors
S particles
Update x_i for each particle i according to v_i
S particles
Initialization of velocity v_i and position x_i of each particle i in the swarm.
no
Stop ?
yes
best solution(s)
S particles
Update the memories of the particles according to their best positions
S particles
Fitness evaluation of the particles

Figure 2.10. *Basic building blocks of PSO algorithms*

2.7.1. *Standard Particle Swarm Algorithm 2007*

There are many variants of PSO [POL 07, ZHA 15]. Among them, we focus on Standard Particle Swarm Optimizer 2007 (SPSO) [CLE 12] mainly for a didactic reason: it slightly modifies the original PSO while significantly improving its efficiency and ease of use. SPSO 2007 was also the best ranked PSO-based approach presented at the Competition on Real-Parameter Single Objective Optimization organized for the IEEE Congress on Evolutionary Computation 2013 [ELA 13]. SPSO 2007, described in Algorithm 16, was considered as the "canonical particle swarm algorithm of today" in 2007 [POL 07]. A more recent version: SPSO 2011, was also submitted to the same contest but it obtained lower results [ZAM 13]. No PSO algorithm competed in the 2014 edition of the contest.

The velocities at iteration t are computed for each particle i according to the following expression:

$$v_{ij}(t+1) \leftarrow w\, v_{ij}(t) + \mathcal{U}[0, c_1](b_{ij}(t) - x_{ij}(t)) + \mathcal{U}[0, c_2](n_{ij}(t) - x_{ij}(t)) \tag{2.11}$$

```
Parameters:
$S$: number of particles in the swarm
$\mathbf{lb}$: vector of lower bounds in the search space
$\mathbf{ub}$: vector of upper bounds in the search space

Result: $\mathbf{g}$: best solution found so far

Variables:
$\mathcal{N} = \{\mathcal{N}_1, ..., \mathcal{N}_S\}$: $\mathcal{N}_i$ is the set of the neighboring particles of particle $i$
$\mathbf{V} = \{\mathbf{v}_1, ..., \mathbf{v}_S\}$: set of the velocity vectors of particles 1 to $S$
$\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_S\}$: set of the position vectors of particles 1 to $S$
$\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_S\}$: set of the best positions of particles 1 to $S$ (memory
of the particles)
$f(\mathbf{x})$: objective or fitness value associated with position $\mathbf{x}$

$w = \frac{1}{2\ln 2}$                                   // inertia coefficient
$c_1 = c_2 = \frac{1}{2} + \ln 2$                        // acceleration factors
$\mathcal{N} \leftarrow$ DefineNeighbors(...)     // possible parameters to define
$\mathbf{V}, \mathbf{X}, \mathbf{B}, \mathbf{g} \leftarrow$ Initialization($S, \mathbf{lb}, \mathbf{ub}$)
repeat
    $\mathbf{V}, \mathbf{X}, \mathbf{B} \leftarrow$ Iteration($\mathbf{V}, \mathbf{X}, \mathbf{B}, w, c_1, c_2$)
    $\mathcal{N} \leftarrow$ DefineNeighbors(...)
    $\mathbf{g} \leftarrow \arg\min_{\mathbf{y} \in \mathbf{B}} f(\mathbf{y})$          // argmax for maximizations
until Stopping criterion satisfied
```

Algorithm 16: Standard Particle Swarm Optimization 2007

$\mathcal{U}[a, b]$ is a random number drawn according to a uniform distribution in the interval $[a, b]$. w is the inertia coefficient while c_1 and c_2 are acceleration factors. These weights are parameters of the algorithm. Note that the accelerations towards the best positions found so far $\mathbf{b}_i$ or $\mathbf{n}_i$ are stronger as their distances from current position $\mathbf{x}_i$ increase. Position $\mathbf{x}_i(t+1)$ is then computed:

$$\mathbf{x}_i(t+1) \leftarrow \mathbf{x}_i(t) + \mathbf{v}_i(t+1)$$

The **Iteration** function (Algorithm 17) implements the body of the loop represented in Figure 2.10. The optional **Confinement** function keeps the particles in a bounded search domain of $\mathbb{R}^d$. This can be useful, especially for constrained optimization. The simplest method consists of giving a constant bad value to the fitness function of particles outside the search domain. In this case, particles are still attracted by the best solutions they have memorized and they come back in the search domain after some iterations. Another approach often used takes back the outside particles to their nearest boundaries and sets their velocities to zero. In this way, the optimizer avoids wasting time exploring outside the search domain. Comparisons of several confinement methods can be found in [HEL 07].

```
$d \leftarrow$ Dimension($\mathbf{x}_1$)
$S \leftarrow |\mathbf{X}|$

for $i = 1$ to $S$ do
    // $\mathbf{n}$: neighbor best position for particle $i$:
    $\mathbf{n} \leftarrow \arg\min_{k \in \mathcal{N}_i} f(\mathbf{b}_k)$          // maximization: argmax
    for $j = 1$ to $d$ do
        // update velocity $\mathbf{v}_i$:
        $v_{ij} \leftarrow w v_{ij} + \mathcal{U}[0, c_1)(b_{ij} - x_{ij}) + \mathcal{U}[0, c_2)(n_j - x_{ij})$
    end
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$                       // update position $\mathbf{x}_i$
    $\mathbf{v}_i, \mathbf{x}_i \leftarrow$ Confinement($\mathbf{v}_i, \mathbf{x}_i$)            // optional

    if $f(\mathbf{x}_i) < f(\mathbf{b}_i)$ then   // maximization: "<" replaced by ">"
        $\mathbf{b}_i \leftarrow \mathbf{x}_i$                  // update the memory of particle $i$
    end
end
return $\mathbf{V}, \mathbf{X}, \mathbf{B}$
```

Algorithm 17: Function Iteration($\mathbf{V}, \mathbf{X}, \mathbf{B}, \mathcal{N}, w, c_1, c_2$)

2.7.2. *The parameters*

Swarm size S for SPSO is suggested to be $\max(40, 10 + 2\sqrt{d})$ [CLE 12]. This value is often appropriate. However, the best value for S that minimizes the evaluation number of the objective function is problem-specific.

Equation [2.11] uses three parameters that determine the particle dynamics. If they are chosen improperly, the velocities can increase indefinitely, which may exceed 10^{100} after some iterations: the swarm explodes. Or the swarm can stagnate: best positions $\mathbf{b}$ or $\mathbf{n}$ are seldom or no longer improved after a number of iterations. From a stability analysis [CLE 02], Clerc and Kennedy have established the following *constriction* relation to avoid the swarm explosion:

$$c_1 + c_2 \leq (w+1)^2 \text{ with } 0.7 \leq w \leq 0.9$$

From a stagnation analysis [CLE 06], the following constant values are suggested for SPSO [CLE 12]:

$$w = \frac{1}{2\ln 2} \approx 0.721$$

$$c_1 = c_2 = \frac{1}{2} + \ln 2 \approx 1.193$$

Greater values for these parameters could accelerate the search of the optimum, but an additional mechanism should then prevent the velocities from exceeding a limit value $V_{\max}$ if the constriction relation is not satisfied. However, this value is problem-specific and there is no exact rule to obtain it. According to Eberhart and Shi [EBE 00], $V_{\max}$ can be set equal to $x_{\max}$, the dynamic range of $\mathbf{x}$ components. The authors also note that the same limit $V_{\max}$ used with the values suggested above for parameters w, c_1 and c_2 can improve the performance of the algorithm for four problems over five from their testbed.

2.7.3. *Neighborhoods*

Each particle i is associated with a subset of neighboring particles $\mathcal{N}_i$, including itself. The neighborhood can be the whole swarm, as with the original PSO [KEN 95]. Small neighborhoods can be preferred to reduce the premature convergence risk. But large neighborhoods may accelerate the convergence towards the optimum. The definition of neighbor set $\mathcal{N}_i$ for each particle i is implemented by function **DefineNeighbors** in algorithm 16. If the neighborhoods are constant during the entire search, the call to **DefineNeighbors** is required only at the initialization step.

2.7.3.1. *The ring topology*

Let S be the size of the swarm, let i be the index of a particle in the swarm, neighborhood $\mathcal{N}_i$ of particle i is defined as $\mathcal{N}_i = \{((i-2+S) \mod S)+1, i, (i \mod S)+1\}$. Thus, $\mathcal{N}_1 = \{S, 1, 2\}$ and $\mathcal{N}_S = \{S-1, S, 1\}$. Such neighborhoods can be modeled by a graph such that each node is associated with a particle and each edge is associated with a pair of neighboring particles. So, the graph can be drawn as a ring. Constant ring topologies are used since the first versions of PSO [EBE 95]. They are still used for their simplicity.

2.7.3.2. *The adaptive random topology*

This topology is proposed for the three versions of SPSO [CLE 12]. Each particle j belongs to K neighborhoods $\mathcal{N}_i$ where i is randomly chosen and also, it belongs to its own neighborhood $\mathcal{N}_j$. K is a parameter of the algorithm, usually set to 3. A neighborhood $\mathcal{N}_i$ can be chosen several times for the same particle j. In this way, for every particle i, $\mathcal{N}_i$ can contain from 1 to S particles with a mean equal to $K+1$.

A new random topology is generated each time an iteration does not improve the fitness value of the best position $\mathbf{g}$ found so far in the swarm. Adaptive topologies contribute to avoid premature convergence.

2.7.4. *Swarm initialization*

Function **Initialization** of Algorithm 16 sets random initial values for positions $\mathbf{x}_i$, the memories $\mathbf{b}_i$ and velocities $\mathbf{v}_i$ of the particles. For instance

$$\mathbf{x}_i \leftarrow \mathbf{b}_i \leftarrow \mathcal{U}[l_i, u_i]$$

$$\mathbf{v}_i \leftarrow (\mathcal{U}[l_i, u_i] - \mathbf{x}_i)/2$$

where l_i and u_i are respectively the lower and upper bounds of component i in a hyper-rectangular domain of $\mathbb{R}^d$.

Simulation results obtained with SPSO 2007 on a set of various objective functions are presented in section 2.8.

2.8. Experiments and performance comparisons

The experiments described in this section aim to identify the best optimization algorithms among CMA-ES, IPOP-CMA-ES, DE, SHADE and SPSO 2007 on sets of test objective functions according to their properties.

2.8.1. *Experiments*

2.8.1.1. *The test problems*

The algorithms presented in previous sections are tested on a suite of 22 non-constrained minimization problems defined in $\mathbb{R}^d$. They are based on ten classical test functions (described in section 2.10) used for the "CEC 2014 Special Session on Real-Parameter Optimization" [LIA 13]. Let f be any of these basic functions and $g(\mathbf{x})$ be the objective function of a test problem. Its search domain is the hypercube $[-100, 100]^d$. $g(\mathbf{x})$ is obtained from f with the following expression:

$$g(\mathbf{x}) = f(a\mathbf{SR}(\mathbf{x} - \mathbf{m})) \times (1 + \alpha|\mathcal{N}(0,1)|) + b$$

where

– $\mathbf{m}$ is the location of the minimum of $g(\mathbf{x})$, which is generated randomly in the domain $[-80, 80]^d$;

– b is the minimum value of g. It is generated at random uniformly from the interval [–1000, 1000];

– α is a noise rate. If $\alpha > 0$, noise is added to the objective function value. $\mathcal{N}(0, 1)$ is a zero-mean, unit-variance Gaussian random value;

– $\mathbf{R}$ is a rotation matrix. If f is not rotated, $\mathbf{R} = \mathbf{I}$. Otherwise, $\mathbf{R}$ is defined as a product of elementary rotation matrices $\mathbf{R}_{kl}$ in the plane defined by axes k and l, for all $k \in \{1, ..., d-1\}$ and $l \in \{k+1, ..., d\}$ (section 2.2.2.6). The angle of each elementary rotation is chosen randomly from a uniform distribution in the interval $[-\pi, \pi]$. The random rotations transform a separable function into a non-separable one;

– **S** is a diagonal matrix, such that $s_{ii} = \kappa^{\frac{1}{2}\frac{i-1}{d-1}}$, for $i \in \{1, ..., d\}$. κ is a given parameter that determines the condition number of **S**. Thus, the smallest coefficient s_{11} is 1, while the largest one is $s_{dd} = \sqrt{\kappa}$. The condition number of **S** is the ratio $s_{dd}/s_{11} = \sqrt{\kappa}$. **S** transforms a well-conditioned problem into an ill-conditioned one if κ is chosen large enough, for instance: $\kappa = 10^6$;

– a is a scaling coefficient that adjusts the search domain to hypercube $[-100, 100]^d$. $a = 1$ except for the following test functions:

	Rosenbrock	Griewank	Rastrigin	Shwefel	Katsuura
a =	2.048/100	6	5.12/100	10	5/100

Parameters $\mathbf{R}$, $\mathbf{m}$ and b are randomly set at the initialization step of each run of the tested algorithms.

The 22 objective functions g are chosen such that the matrix **R** is randomly generated for 10 of them and 12 of them are ill-conditioned such as functions **Discus** and **Bent Cigar**, or by setting $\kappa = 10^6$. Note that **Discus** or **Bent Cigar** are quadratic functions, whose the condition number of the Hessian matrix is 10^6, from the definition of these functions (section 2.10). κ is set to 10^6 for **Weierstrass** and **Griewank** functions to better decorrelate the ill-conditioned and multimodal categories. **Noisy_Ellipse_R** is the only objective function with noise: $\alpha = 0.4$. κ is set to 10^4 for this function because $\kappa = 10^6$ makes its optimum too hard to locate, whatever the method used. $\alpha = 0$ for all other functions. Functions g are listed in Table 2.2.

2.8.1.2. *Free parameters of the algorithms*

CMA-ES requires the values of two free parameters: initial vector $\mathbf{m}(0)$ and the value of $\sigma(0)$. $\mathbf{m}(0)$ is chosen equal to 0, as the center of the search domain $[-100, 100]^d$, and $\sigma(0) = 100$ ensures a good diversity for the random solutions $\mathbf{x}_i$ in the search domain at generation 0. Choosing $\mathbf{m} = \mathbf{0}$ does not introduce a bias because the optima are randomly set in domain $[-80, 80]^d$. The other parameters are initialized by Algorithm 6 (p. 56).

The same parameter values as CMA-ES are chosen for IPOP-CMA-ES. In addition, the increasing factor of the population size is set to 2.

The particle swarm optimizer SPSO-2007 only needs a swarm size, which has been set to 40, in accordance with the suggestion of the algorithm designers (section 2.7.2). Preliminary tests confirmed this value.

g	f (see section 2.10)	κ	Rotated	ill-conditioned	separable	multi-modal
Sphere	Sphere	1		no	yes	no
Ellipse	Sphere	10^6		yes	yes	no
Ellipse_R	Sphere	10^6	$\surd$	yes	no	no
Noisy_Ellipse_R	Sphere	10^4	$\surd$	yes	no	no
BentCigar	BentCigar	1		yes	yes	no
BentCigar_R	BentCigar	1	$\surd$	yes	no	no
Discus	Discus	1		yes	yes	no
Discus_R	Discus	1	$\surd$	yes	no	no
Rosenbrock	Rosenbrock	1		yes	no	no
Rosenbrock_R	Rosenbrock	1	$\surd$	yes	no	no
Ackley	Ackley	1		no	yes	yes
Ackley_R	Ackley	1	$\surd$	no	no	yes
Weierstrass_I	Weierstrass	10^6		yes	yes	yes
Weierstrass_IR	Weierstrass	10^6	$\surd$	yes	no	yes
Griewank_I	Griewank	10^6		yes	no	yes
Griewank_IR	Griewank	10^6	$\surd$	yes	no	yes
Rastrigin	Rastrigin	1		no	yes	yes
Rastrigin_R	Rastrigin	1	$\surd$	no	no	yes
Schwefel	Schwefel	1		no	yes	yes
Schwefel_R	Schwefel	1	$\surd$	no	no	yes
Katsuura	Katsuura	1		no	yes	yes
Katsuura_R	Katsuura	1	$\surd$	no	no	yes

Table 2.2. *Set of test functions g and their parameters. The properties of g are given in the three last columns. For instance, the table shows that* **Griewank_IR** *is obtained from basic function f "Griewank", a random rotation defined by matrix $\mathbf{R}$ and parameter $\kappa = 10^6$ for matrix $\mathbf{S}$. The table indicates that* **Griewank_IR** *is ill-conditioned, non-separable and multimodal*

SHADE needs the choice of population size N, history size S_M and size S_A for the external archive. $N = 100$ is often suggested in the literature [BRE 06] [TAN 13]. Preliminary experiments have shown this choice is relevant. The same value is given to the other parameters: $S_M = S_A = 100$, as in [TAN 13].

The canonical Differential Evolution (Algorithm 7, p. 57) requires population size N, scaling factor F, crossover rate CR, the choice of the mutation variant and the crossover variant. These parameters have been chosen close to those of SHADE, with the help of some light preliminary experiments to confirm that they are relevant. Thus, the following parameter values are chosen for DE: $N = 100$, $F = \mathcal{C}_{[0,1]}(0.7, 0.2)$ is a random number following a truncated Cauchy distribution in the interval $[0, 1]$, with median equal to 0.7 and scale equal to 0.2, CR $= 1$ with the *DE/current-to-best/1* mutation variant. $CR = 1$ disables crossover, so the choice of a crossover variant is useless. Indeed, the standard crossover variants (binomial or exponential) are harmful to the convergence speed for non-separable objective functions (section 2.5.3.1). *DE/current-to-best/1* mutation is chosen because it is similar to the mutation operator used with SHADE. Moreover, it behaves partly as a linear BLX-0 crossover (section 2.5.2), more compatible with non-separable functions. By this choice, we clearly want to improve the optimization of non-separable functions.

Obviously, this way of choosing parameter values is non-optimal, somewhat arbitrary. But this choice avoids possibly heavy preliminary experiments to tune parameters, involving large numbers of objective function evaluations. In fact, tuning parameters is an optimization problem by itself with at least the following objectives: finding the best solutions while minimizing the number of function evaluations to solve a class of problems [EIB 11]. However, by the fact of its operation, the tuning process degrades its own objective function values. Thus, for instance, let us consider a parameter tuning that needs 10^7 function evaluations to obtain a good set of parameters, able to find a near optimal solution with less than 10^4 evaluations. This tuning process will be useful if the set of parameters is reused many times, for instance in industrial products. But, it is useless if this good set of parameters is used only once.

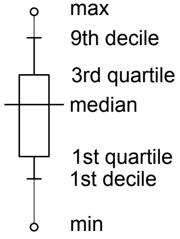
2.8.1.3. *Experiment parameters*

For all objective functions, the tested algorithms stop when $|g(\mathbf{x}) - b| = \varepsilon \leq 10^{-8}$ or when the number of objective function evaluations (FEs) reaches 10^6. The performances are then measured by pairs $(\mathrm{FEs}, \varepsilon)$.

Evolutionary algorithms are stochastic methods and no significant information about their efficiencies can be drawn from only one run. Each performance result presented in the following section is obtained from 100 independent runs with the same parameters for each objective function g. A performance result is thus a random sample of 100 pairs $(\mathrm{FEs}, \varepsilon)$.

2.8.2. *Results*

This section presents the experimental results obtained with the CMA-ES, Differential Evolution and Particle Swarm Optimization algorithms described in previous sections. Each graph represents the performance results for a given algorithm, such as Figure 2.11, for instance. The samples $(\text{FEs}, \varepsilon)$ are presented as two box plots combined in a 3D graph. They use whisker boxes as described on the right-hand side of this paragraph.



Boxes in the "vertical" plane represent values and dispersions of FEs samples for the tests functions when $\varepsilon < 10^{-8}$. Boxes in the "horizontal" plane in parallel perspective represent values and dispersions of errors ε as FEs $\geq 10^6$. The percentages below some FEs boxes indicate the rate of runs in the result samples with $\varepsilon < 10^{-8}$. For instance, 87% in the column "Rosenbrock" in Figure 2.11 indicates that 87% of runs reached an error less than 10^{-8} for test function **Rosenbrock**. Consequently, the corresponding error box in the "horizontal" plane is built with 13% of runs. No percentage given under FEs boxes means that 100% of runs reached an error less than the threshold value 10^{-8}.

For each test function, the performance medians obtained with the best algorithms is also represented in graphs as circles. The first three letters of their names are given in the lower part of the graphs. Thus, Figure 2.11 indicates that algorithm SHADE is the most efficient one to find the optimum of function **Ellipse**: the evaluation number is near 10^5 to reach $\varepsilon < 10^{-8}$ with SHADE and in the interval $[4 \cdot 10^5, 5 \cdot 10^5]$ with CMA-ES. Several methods may be referred to as the best ones for some test functions if their performance medians differ by less than 5%, such as CMA-ES, IPOP-CMA-ES and SHADE for problem **Rosenbrock**.

At last, the bottom sides of gray bars in the "Error" plane of the graphs show the median performance achieved by a sample of 100 uniform distribution random searches with one million objective evaluations for each of them (Algorithm 18). For instance, Figure 2.12 shows that IPOP-CMA-ES has the same median performance as random searches for problems **Katsuura** and **Katsuura_R**.

```
Parameter: $d$: dimension of the search space
Result: b: best solution found so far

$\mathbf{b} \leftarrow 0$
for $i = 1$ to $10^6$ do
    $\mathbf{x} \leftarrow \mathcal{U}[-100, 100]^d$
    // x is a uniform random vector in box $[-100, 100]^d$
    if $f(\mathbf{x}) < f(\mathbf{b})$ then          // $f$ is the objective function
        $\mathbf{b} \leftarrow \mathbf{x}$
    end
end
```

Algorithm 18: Random search

The performance data for each algorithm and all test problems shown in Figures 2.11 to 2.15 are summarized as algorithm rankings in Table 2.3. They are drawn up on the basis of performance median comparisons.

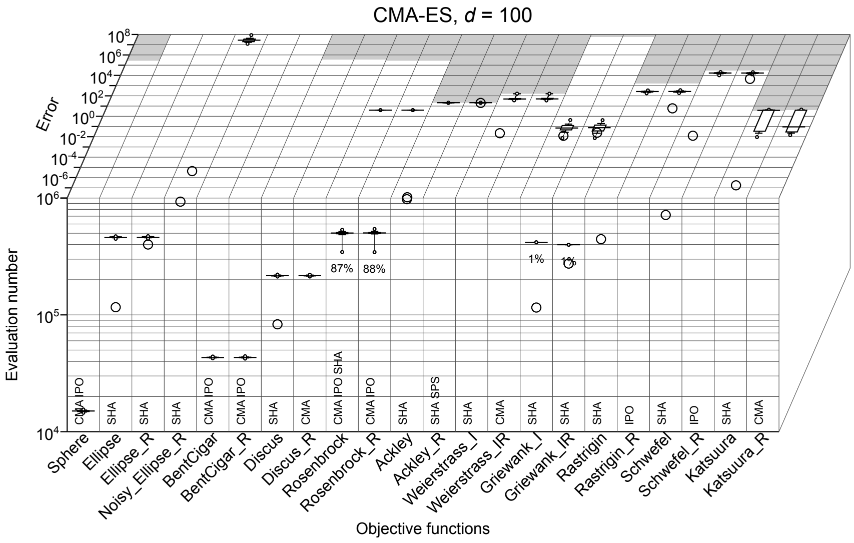


Figure 2.11. *Performance of the CMA-ES algorithm on a set of 22 objective functions in dimension* $d = 100$

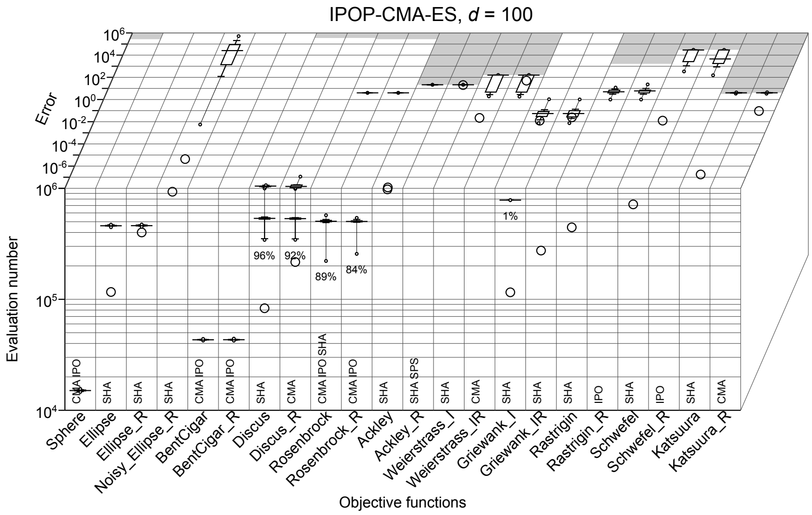


Figure 2.12. *Performance of the IPOP-CMA-ES algorithm on a set of 22 objective functions in dimension $d = 100$*

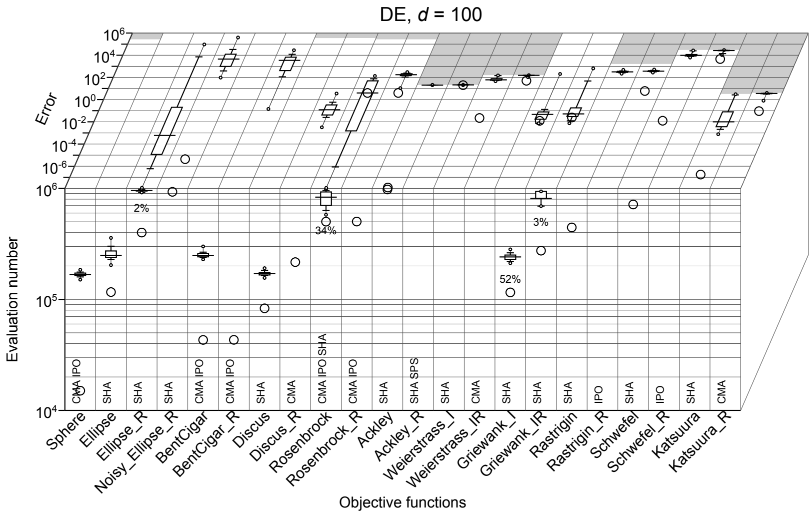


Figure 2.13. *Performance of the canonical Differential Evolution method on a set of 22 objective functions in dimension $d = 100$*

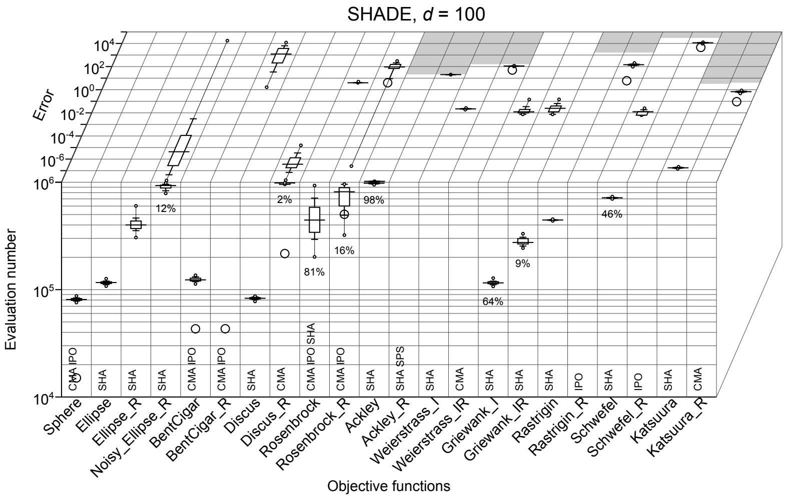


Figure 2.14. *Performance of the SHADE-1 method on a set of 22 objective functions in dimension $d = 100$*

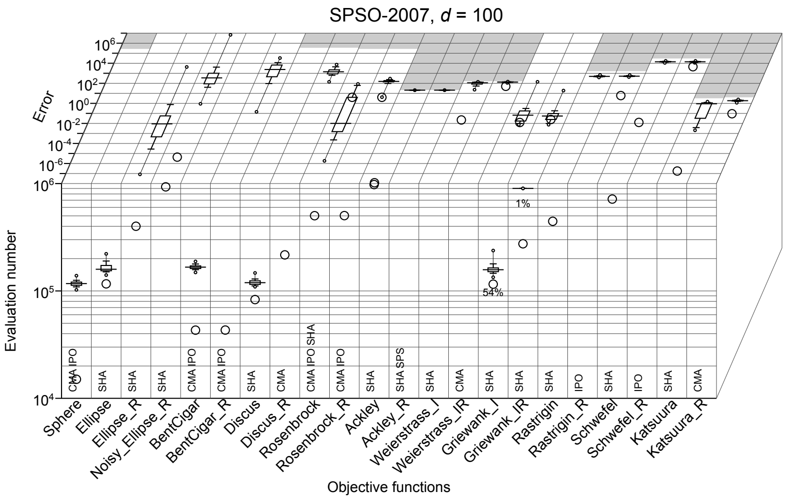


Figure 2.15. *Performance of the "Standard Particle Swarm Optimization 2007" method on a set of 22 objective functions in dimension $d = 100$*

2.8.3. *Discussion*

Table 2.3 shows immediately that SHADE is the best algorithm for 13 test problems over 22. CMA-ES, which is the best one for eight test problems, comes in second place. The third place is given to IPOP-CMA-ES, which is the best algorithm for seven problems. Table 2.3 indicates that SPSO reaches a best solution only for the **Ackley_R** function. In fact, Figures 2.11–2.15 show that no algorithm is significantly better than random explorations of the search domain for function **Ackley_R**. Thus, SPSO is weakly dominated by other algorithms for all problems in the test set. Finally, the DE algorithm is dominated by other methods for all test problems.

Objective functions	CMA-ES	IPOP-CMA-ES	DE	SHADE	SPSO 2007
Sphere	1	1	5	3	4
Ellipse	5	5	3	1	2
Ellipse_R	2	2	4	1	5
Noisy_Ellipse_R	5	4	3	1	2
BentCigar	1	1	5	3	4
BentCigar_R	1	1	5	3	4
Discus	4	5	3	1	2
Discus_R	1	2	4	3	5
Rosenbrock	1	1	4	1	5
Rosenbrock_R	1	1	5	3	4
Ackley	2	2	2	1	2
Ackley_R	3	3	3	1	1
Weierstrass_I	2	5	3	1	4
Weierstrass_IR	1	4	4	2	3
Griewank_I	5	4	3	1	2
Griewank_IR	5	2	2	1	2
Rastrigin	3	2	4	1	5
Rastrigin_R	3	1	4	2	5
Schwefel	4	5	2	1	3
Schwefel_R	4	1	5	2	3
Katsuura	4	4	2	1	3
Katsuura_R	1	5	4	2	3

Table 2.3. *Algorithm rankings for each test function*

However, these scores are not highly significant because they depend on the composition of the set of test problems. It is interesting to assess algorithm performances in regard to the following properties:

– ill-conditioned problems;

– separable problems;

– multimodal problems.

These properties are given by Table 2.2 for each problem. We consider that an algorithm succeeds for a given objective function if the median error obtained after 100 runs is significantly less than the median error given by simple random searches. For the present testbed, we define “significantly less than” as “less than one tenth of”. This threshold is chosen because it can be easily visualized from the logarithmic grid in Figures 2.11–2.15. Table 2.4 indicates for each algorithm its success rates in regard to the objective function properties. This table is built from Table 2.2 and the above-mentioned figures.

			separable, multimodal			
			no, no	no, yes	yes, yes	yes, no
ill-conditioned	no		0 function	4 functions	4 functions	1 function
		CMA		1/4	0/4	success for all the algorithms
		IPO		2/4	1/4	
		DE		0/4	1/4	
		SHA		1/4	success	
		SPS		0/4	0/4	
	yes		6 functions	3 functions	1 function	3 functions
		CMA	success for all the algorithms	2/3	0/1	success for all the algorithms
		IPO		2/3	0/1	
		DE		2/3	0/1	
		SHA		2/3	success	
		SPS		2/3	0/1	

Table 2.4. *Success rates; for instance, in the category of “well-conditioned, non-separable and multimodal functions” (row “no”, column “no, yes”), the table shows that IPOP-CMA-ES succeeds for 2 functions over the 4 functions of the category. “success” means that an algorithm succeeds for all the functions of the category*

Table 2.4 clearly shows that the biggest challenge is multimodality: CMA-ES fails to be significantly better than the random search for nine multimodal functions over 12. IPOP-CMA-ES improves the score with seven failures over 12 functions. SHADE is the champion with “only” four failures over 12 functions. Among them, however, the good score of SHADE is due to separable functions.

Table 2.5 classifies the best algorithms, i.e. those that are ranked at the first place in Table 2.3, according to the test function properties. Results obtained for problems **Weierstrass_IR** and **Ackley_R** are not taken into account in Table 2.5 because all the algorithms fail for these functions. For instance, in the group of ill-conditioned, non-separable and mutimodal test functions, “SHA 2/2” means that SHADE is the best algorithm for two test problems among the two problems identified with Table 2.2 as: **Griewank_I** and **Griewank_IR**, but not **Weierstrass_IR** for which all the algorithms fail.

		separable, multimodal			
		no, no	no, yes	yes, yes	yes, no
ill-conditioned	no		CMA 1/3 **IPO 2/3**	**SHA 4/4**	**CMA 1/1** **IPO 1/1**
	yes	**CMA 4/6** IPO 3/6 SHA 3/6	**SHA 2/2**	**SHA 1/1**	CMA 1/3 IPO 1/3 **SHA 2/3**

Table 2.5. *Classification of the best algorithms according to the properties of the test functions. For instance, in the 2nd row, first column, the table indicates that CMA-ES is the best algorithm for 4 problems among the 6 ill-conditioned, non-separable and unimodal problems of the test set*

Table 2.5 shows that the best results with SHADE when compared to other algorithms are obtained with 100% of multimodal separable functions in the test set. SHADE is also mostly the best algorithm for other categories of functions such as the ill-conditioned, separable and unimodal functions. This is also surprisingly the case for the ill-conditioned, non-separable and multimodal functions of the test set. Indeed, the DE algorithms are known to be less-efficient on rotated test problems [HAN 04]. Thus, the archive and history of the SHADE variant seem to improve this point. SHADE is also by far the best method to find a near optimal solution of the noisy rotated ellipse function with a median error less than 10^{-5}, as shown in Figures 2.11–2.15. More studies are, however, required with a larger set of noisy test functions to better characterize this ability.

Otherwise, CMA-ES yields the best results for non-separable unimodal functions with four successes over six test functions. However, CMA-ES has not the majority of successes among the best-ranked algorithms in other categories of test problems, except for the easy **Sphere** problem (well conditioned, separable, unimodal). Figures 2.11 and 2.12 show that the performance is the same for rotated objective functions and the non-rotated versions. The rotation invariance is a feature of the CMA-ES [HAN 96].

The best results specific to IPOP-CMA-ES are obtained with two of the three well-conditioned and non-separable multimodal functions. All the other best results given by this algorithm are in fact identical to those of CMA-ES. In this case, the restart mechanism of IPOP-CMA-ES is not activated during runs.

2.9. Conclusion

This chapter has described five efficient evolutionary methods to solve non-constrained, nonlinear continuous optimization problems:

– the Covariance Matrix Adaptation Evolution Strategy, which is rotation-invariant, thus helping to find optima of non-separable objective functions;

– the Covariance Matrix Adaptation Evolution Strategy with Increasing Population size, which aims to improve the efficiency of CMA-ES for multimodal objective functions;

– the Differential Evolution (DE) algorithm, which is a simple and quite efficient algorithm, is able to deal with ill-conditioned objective functions, with few parameters compared to older evolutionary approaches such as genetic algorithms;

– the Success-History based Adaptive Differential Evolution, which is a powerful variant of DE algorithms;

– the Standard Particle Swarm Optimizer 2007, which is very close to the original algorithm while improving significantly its performances and ease of use.

These five methods are recognized as being among the most efficient or the most popular ones to date. The experiments presented in this chapter show that, among these algorithms, there is no universal method that gives the best results for all the objective functions of the test set. This suggests that specific mechanisms at the origin of the good performances of these algorithms could be combined to design more robust methods. Thus, a recent variant of SHADE has replaced the standard binomial crossover of the original algorithm by a rotationally-invariant, eigenvector-based crossover to improve its performance with non-separable objective functions [GUO 15].

It can be noted from the experimental results that canonical DE and SPSO 2007 are dominated by CMA-ES, IPOP-CMA-ES and SHADE, whatever the objective functions are. In fact, this observation opposes the adaptive methods to those that are not. Indeed, it seems reasonable that self-adaptive methods able to find near-optimal values for critical parameters along an evolution perform better than non-adaptive methods. Often, parameter control mechanisms introduce additional meta-parameters, such as the history size for SHADE or the increasing factor of the population size for IPOP-CMA-ES. So, self-adaptive algorithms should be designed such that their performances are relatively insensitive to these meta-parameters around robust values proposed by the designers [KAR 15]. Experiments have shown that this is the case for the three best algorithms presented in this chapter: CMA-ES, IPOP-CMA-ES and SHADE.

2.10. Appendix: set of basic objective functions used for the experiments

These functions were used for the CEC 2014 “special session and competition on single objective real-parameter numerical optimization” [LIA 13]. The global minimum for each function is $f(\mathbf{0}) = 0$. d is the dimension of the search space in the expressions given below.

– Sphere function:

$$f_1(\mathbf{x}) = \sum_{i=1}^{d} x_i^2$$

Properties: convex, separable, well-conditioned.

– Bent Cigar function:

$$f_2(\mathbf{x}) = x_1^2 + 10^6 \sum_{i=2}^{d} x_i^2$$

Properties: convex, separable, ill-conditioned.

– Discus function:

$$f_3(\mathbf{x}) = 10^6 x_1^2 + \sum_{i=2}^{d} x_i^2$$

Properties: convex, separable, ill-conditioned.

– Shifted Rosenbrock's function:

$$f_4(\mathbf{x}) = \sum_{i=1}^{d-1} \left(100(y_i^2 - y_{i+1})^2 + (y_i - 1)^2\right) \quad \text{with} \quad y_i = x_i + 1$$

Properties: non-convex, unimodal, non-separable, ill-conditioned.

– Ackley's function:

$$f_5(\mathbf{x}) = -20 \exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d} \cos(2\pi x_i)\right) + 20 + e$$

Properties: multimodal, separable.

– Weierstrass function:

$$f_6(\mathbf{x}) = \sum_{i=1}^{d} w_K(x_i) - d\, w_K(0) \quad \text{with} \quad w_K(x) = \sum_{k=0}^{K} a^k \cos(2\pi b^k (x + 0.5))$$

and $a = 0.5$, $b = 3$, $K = 20$

Properties: multimodal, separable, continuous, differentiable nowhere as $K \to \infty$.

– Griewank's function:

$$f_7(\mathbf{x}) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Properties: multimodal, non-separable.

– Rastrigin's function:

$$f_8(\mathbf{x}) = \sum_{i=1}^{d} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$$

Properties: multimodal, separable.

– Modified Schwefel's function:

$$f_9(\mathbf{x}) = 418.9828872724338 \times d - \sum_{i=1}^{d} g(x_i + 420.9687462275036)$$

$$g(x) = \begin{cases} x\sin(\sqrt{|x|}) & \text{if } |x| \leq 500 \\ y\sin(\sqrt{|y|}) - \frac{(x-500)^2}{10000\ d} & \text{if } x > 500 \\ z\sin(\sqrt{|z|}) - \frac{(x+500)^2}{10000\ d} & \text{if } x < -500 \end{cases} \quad \text{with} \begin{cases} y = 500 - x \bmod 500 \\ z = |x| \bmod 500 - 50 \end{cases}$$

Properties: multimodal, separable.

– Katsuura's function:

$$f_{10}(\mathbf{x}) = \frac{10}{d^2} \prod_{i=1}^{d} \left(1 + i \sum_{j=1}^{K} \frac{|2^j x_i - \lfloor 2^j x_i + 0.5 \rfloor|}{2^j}\right)^{\frac{10}{d^{1.2}}} - \frac{10}{d^2}$$

with $K = 32$

Properties: multimodal, separable, continuous, differentiable nowhere as $K \to \infty$.

3

Constrained Continuous Evolutionary Optimization

3.1. Introduction

Constrained Evolutionary Optimization (CEO) is a somewhat different approach from unconstrained optimization, since the goal is not to find the global optimum of the objective function but to obtain the best solution that satisfies the constraints. However, there is no canonical evolutionary algorithm (EA) able to take the constraints into account, since the regular search operators are "blind" to constraints. Thus, a wide range of approaches and heuristic algorithms were proposed to extend EA for the constrained nonlinear optimization.

This chapter, after formally presenting the constrained optimization, describes the different approaches for handling constraints in evolutionary computation and presents a survey of some of the methods for each approach. It also discusses the effectiveness of some of these approaches and provides the results of a comparison between three of them.

3.1.1. *The problem with Constrained Evolutionary Optimization*

In the field of continuous optimization, constraints are expressed as a set of relations that the variables of the objective function must satisfy. These relations are usually presented as equalities and inequalities that may be very hard to deal with. The general nonlinear parameter optimization problem is then defined as:

$$\text{optimize } f(\mathbf{x}), \mathbf{x} = (x_1, \dots, x_n) \in \mathcal{F} \subseteq \mathcal{S} \subseteq \mathbb{R}^n,$$

subject to:

$$\begin{cases} g_i(\mathbf{x}) \leq 0, \ \forall i \in \{1, ..., q\} \text{ (inequality constraints)} \\ h_j(\mathbf{x}) = 0, \forall j \in \{q+1, ..., m\} \text{ (equality constraints)} \end{cases}$$

where $\mathcal{F}$ is the *feasible region* where f, g_i and h_j are real-valued functions on $\mathbb{R}^n$; $\mathcal{S}$ is the *search space*; q is the number of inequality constraints and $m - q$ is the number of equality constraints (in both cases, constraints can be linear or nonlinear). In the following, we consider only minimization problems. A maximization problem can be transformed into a minimization problem by inverting the objective function, $f(\mathbf{x})$.

The search space, $\mathcal{S} \subseteq \mathbb{R}^n$, is defined by the lower and upper bounds of the solution vectors for each of these components:

$$l(i) \leq x_i \leq u(i) \ \forall i \in \{1, ..., n\}$$

The satisfaction of the set of constraints g_j, $(j \in \{1, ..., q\}$ and h_j, $(j \in \{q+1, ..., m\})$ defines the feasible region, $\mathcal{F}$. Any solution belonging to $\mathcal{F}$ is a feasible solution, otherwise it is unfeasible. The search for the feasible optimum solution is more difficult when the feasible space size is small and/or its shape is complex (e.g. $\mathcal{F}$ is a set of dispersed small areas).

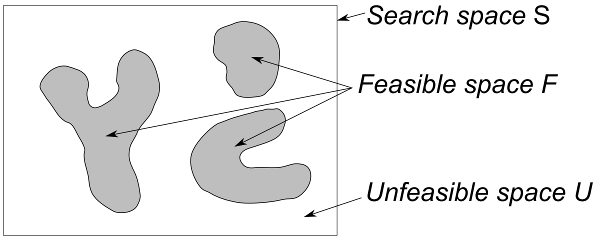


Figure 3.1. *A search space $\mathcal{S}$ and its feasible and unfeasible parts*

The ratio $|\mathcal{F}|/|\mathcal{S}|$ of feasible points in $\mathcal{S}$ can be used as a measure of the difficulty of the problem [MIC 96c]. The search for the optimum is often easier when it is inside the feasible region than when it is on its boundary. The latter case arises when one (or several) constraint(s) of the problem is (are) active at the optimum solution. This is often the case in real-world problems. Thus, constrained optimization problems require an exhaustive search of the feasible domain [HAM 00, MIC 96c].

3.1.2. *Taxonomy*

There is no dedicated evolutionary method to determine the global optimum of a constrained problem. The main question is: how to deal with unfeasible solutions? To reply to this question, two strategies are formed: the first considers only the feasible individuals, and therefore the objective function is computed only in the feasible domain. The second considers all individuals in the search space but requires a special handling of the unfeasible individuals.

Choosing the right strategy depends on the nature of the problem, for example, whether the objective function is defined in the unfeasible domain or not.

A multitude of methods have been proposed for the two strategies that can be classified into six main categories:

– penalty methods;

– methods based on the assumption of superiority of feasible individuals;

– methods evolving in the feasible region;

– methods using multi-objective optimization techniques;

– methods using parallel population approaches;

– hybrid methods.

Each category is divided into several sub-categories. We propose in Figure 3.2 a taxonomy of the constraint-handling approaches for evolutionary optimization.

Let us consider a simple flowchart of an Evolutionary Algorithm which summarizes the main steps of the evolution process (Figure 3.3). Introducing a technique to handle constraints in an EA means modifying one or several steps in the corresponding scheme.

The simplest way to adapt an EA to the constrained optimization without needing to re-implement the three basic evolutionary procedures (i.e. selection, crossover, mutation) is the penalization of unfeasible solutions. To introduce the penalization, only the evaluation step is concerned. Indeed, the fitness function is redefined in order to differentiate between feasible and unfeasible individuals. This approach is detailed in section 3.2 with a presentation of some known methods in this category.

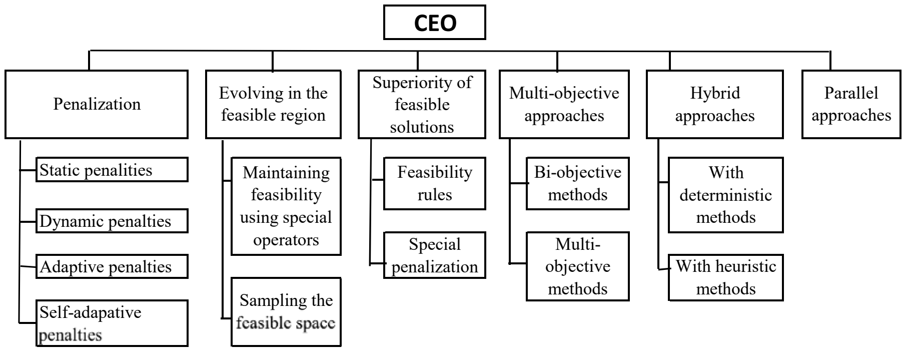


Figure 3.2. *Taxonomy of the constraint-handling techniques with EAs.*

Another way to handle constraints is to assume that any feasible solution is better than any unfeasible solution. This superiority might be expressed by modifying the evaluation function in a similar way to the penalization approach and/or by introducing new rules in the selection process, as illustrated in Figure 3.3. In the first case, only the evaluation step is concerned like for the penalization approach; however, in the second case, the selection procedure must be re-implemented. This is the case for the methods of the second category detailed in section 3.3.

If the objective function is not defined in the unfeasible domain, many methods in the third category propose to evolve the population in the feasible region, either by sampling the feasible region before the evolution or by using a special operator to maintain the feasibility of the population (Figure 3.3). The second case requires special evolutionary operators able to keep individuals within the feasible space. The methods in this category are more complex. Section 3.4.1 presents some of them.

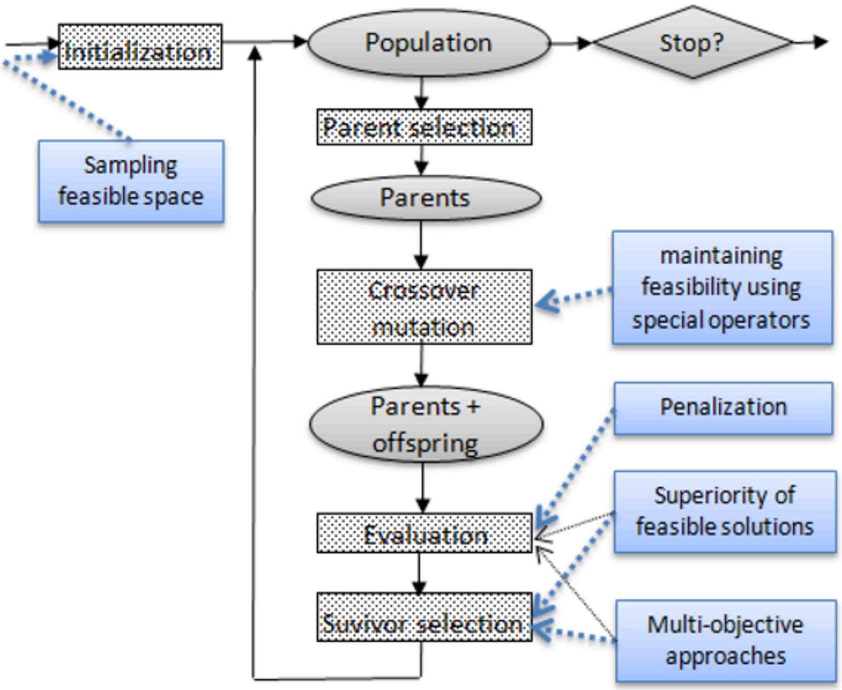


Figure 3.3. *Evolutionary steps affected by the constraint-handling approaches for CEO*

An attractive approach to handling constraints in the CEO is the use of multi-objective techniques. A great number of methods were proposed in this category, implementing bi-objective or multi-objective mechanisms. Most of them could be implemented by modifying only the environmental selection mechanism. Other methods need to redefine parts such as the evaluation procedure or even the genetic operators (see Figure 3.3).

Some recent methods implement several constraint-handling solutions in a parallel way or combine the EA with other optimization methods. They are presented in section 3.6. These approaches are not illustrated in Figure 3.3 since they do not use the regular EA.

Note that it is not possible to present all available methods in each category or sub-category. The papers [MIC 96c, COE 02a, MEZ 11] present a comprehensive survey of the constraint-handling techniques with Evolutionary Algorithms published at different periods (1996, 2002 and 2011, respectively). In this chapter, we present the basics and some reference methods for each category.

3.2. Penalization

Most constraint-handling methods are based on the concept of penalty functions, which penalize unfeasible solutions by adding a positive quantity to the objective function f (when the goal is to minimize f), in order to decrease the quality of such unfeasible individuals. The initial constrained problem is then converted into an unconstrained problem as follows:

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) + p(\mathbf{x}) \\ &p(\mathbf{x}) \begin{cases} = 0 & \text{if } \mathbf{x} \in \mathcal{F} \\ > 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.1}$$

where $p(\mathbf{x})$ is the penalty function.

The design of the penalty function, $p(\mathbf{x})$, is the main difficulty in penalty methods. Several techniques with different approaches and different heuristics have been proposed. The most popular approaches use measures of the constraint violations:

$$p(\mathbf{x}) = F(\sum_{j=1}^{m} \alpha_j v_j^{\beta}(\mathbf{x}))\,, \tag{3.2}$$

where F is an increasing function, the positive real numbers $\alpha_j; j = 1 \cdots m$ are called the penalty coefficients, β is a parameter (often equal to 2) and $v_j(\mathbf{x})$ is the j^{th} constraint violation (distance to the feasible region) defined as follows:

$$v_j(\mathbf{x}) = \begin{cases} \max(0, g_j(\mathbf{x})), & j \in \{1, ..., q\} \text{ (inequality constraints)} \\ |h_j(\mathbf{x})|, & j \in \{q+1, ..., m\} \text{ (equality constraints)} \end{cases} \tag{3.3}$$

The sum $V(\mathbf{x})$ of constraint violation, $v_j(\mathbf{x})$, gives an estimation of the total violation by solution $\mathbf{x}$:

$$V(\mathbf{x}) = \sum_{j=1}^{m} v_j(\mathbf{x}) \tag{3.4}$$

Using the constraint violation measure in the penalty function helps to distinguish the unfeasible individuals. Indeed, those having a high violation measure are penalized more than those with low violations, which can guide the search for the feasible space.

However, this measure is generally insufficient to set the penalty function, especially if the difference between the objective function values and the violation measures is very high (e.g. with an objective function in the order of 10^5 and a violation measure in the order of 10^{-1}). Penalty coefficients are then used to adjust the quantity to be added to the objective function. The main difficulty is to determine the appropriate value for each coefficient. If the penalty is too high or too low, then the problem might be even more difficult for an EA. With low values for penalty coefficients, the algorithm may produce much more unfeasible solutions than feasible solutions and then a lot of time will be spent exploring the unfeasible region. Otherwise, a large penalty discourages the exploration of the unfeasible region and may increase the risk of premature convergence, especially if $\mathcal{F}$ is not convex or is disjoint.

In order to investigate the effect of the penalty coefficient on the performance of GAs, let us take a simple example. We consider the following problem:

$$\text{minimize } f(\mathbf{x}) = 0.5\sin(x) + 5, \text{ with } 0 \leq x \leq 10$$

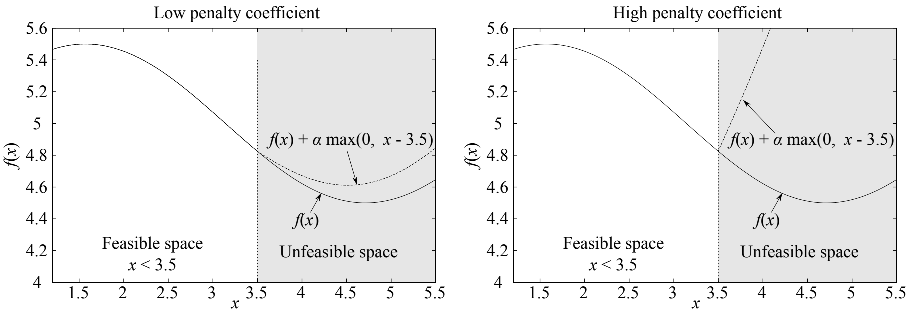
The problem is subject to a single constraint: $x \leq 3.5$. The optimum for this problem is $\mathbf{x}^* = 3.5$. The penalty function is defined as follows:

$$p(\mathbf{x}) = \alpha \max(0, x - 3.5)$$

Despite the simplicity of the problem, an EA may fail in its resolution if the value of α is not adequate. Figure 3.4(a) shows the curve of the objective function $f(\mathbf{x})$ and that of the evaluation function (fitness) $f(\mathbf{x}) + p(\mathbf{x})$ on the unfeasible region, with $\alpha = 0.1$. Clearly, the minimum for the evaluation function is $\mathbf{x}^* = 4.5$, which is an unfeasible solution. Thus, a penalty factor equal to 0.1 is too low to guarantee the feasibility of the returned solutions.

To overcome this problem, it is necessary to use a higher penalty coefficient. On the other hand, a very large value results in a sudden rejection of all unfeasible solutions in the population from the start of the evolution. In this case, the penalization may forbid any shortcuts across the unfeasible region and restrict the search in some parts of the feasible domain, which may lead to the failure of the method.

Figure 3.4(b) shows the curve of $f(\mathbf{x})$ and that of $f(\mathbf{x}) + p(\mathbf{x})$ in the unfeasible area, with $\alpha = 1.7$. The slope of the curve of the fitness function for $x \leq 3.5$ is very high, which induces a weak production of solutions located in this part of the search space.



a) $\alpha = 0.1$ b) $\alpha = 1.7$

Figure 3.4. *Curves of the objective function* $f(x) = 0.5\sin(x) + 5$ *and the fitness function* $f(x) + \alpha\max(0, x - 3.5)$*, with* $\alpha = 0.1$ *a) and* $\alpha = 1.7$ *b)*

Since the optimum is on the boundary, the algorithm will have some difficulties generating accurate solutions close to its position. In this case, there is always more of a chance to locate the optimum when the surrounding areas of the boundary are explored from both sides than from one side only.

Otherwise, if the feasible space, $\mathcal{F}$, is non-convex or disjoint, the presence of unfeasible solutions in the population improves the exploration capacity of the algorithm by spreading the population throughout the different parts of $\mathcal{F}$.

The choice of the penalization method must take into account the topological properties of the feasible space, the number of active constraints at the optimum, the ratio between the size of $\mathcal{F}$ and $\mathcal{S}$, as well as the type of objective function and constraints.

We can distinguish four approaches to define the penalty function: the static approach, the dynamic approach, the adaptive approach and the self-adaptive approach. With the static approach, the penalty coefficients are parameters of the algorithm and their values are constant during the evolution. With the other approaches, their values are modified throughout the evolution, according to a pre-defined pattern for the dynamic approach and depending on the historical and/or current status of the population for the adaptive and self-adaptive approaches. Note that the "death penalty" method proposed by [BAC 91] is also considered as a penalization technique with infinite penalty ($p(\mathbf{x}) = +\infty$). This method simply rejects unfeasible solutions from the population. However, this strategy performs poorly with benchmarks and real-world problems, and it is seldom used.

The first method using static penalties is proposed by Homaifer *et al.* [HOM 94] and uses a large set of penalty coefficients defined for different violation rates for each constraint. This large number of parameters to define makes this technique perform weakly.

The dynamic penalty function is generally an increasing function of the generation number in order to ensure the feasibility of solutions at the end of the evolution, but the modification scheme is not easy to define and depends on the problem. As regards to the adaptive and self-adaptive methods, they modify the penalty coefficients according to information extracted from the population, essentially the feasibility of a certain proportion of the population, its best solution, or the distance to the feasible region.

3.2.1. *Static penalties*

The static penalty methods use user-defined values for penalty coefficients α_j. Choosing values for these coefficients might be problematic because of the risks of over/under-penalization discussed above. In an attempt to avoid this risk, Homaifar, Lai and Qi proposed in 1194 a sophisticated method which defines for each constraint a family of violation intervals, and then, for each interval, an appropriate penalty coefficient [HOM 94]. The method can be summarized in the following steps:

– for each constraint, create a number (l) of violation rates;

– for each constraint and for each violation rate, define a penalty coefficient $\alpha_{ij} (i = 1, 2, \cdots, l,\ j = 1, 2, \cdots, m)$;

– the coefficients having the biggest values are allocated to the highest violation rates;

– the initial population is generated randomly without taking into account the feasibility of individuals;

– during the evolution, each individual is evaluated using the following formula:

$$eval(\mathbf{x}) = f(\mathbf{x}) + \sum_{j=1}^{m} \alpha_{ij} v_j^2(\mathbf{x})$$

The main drawback of this method is the number of parameters to be set. Indeed, for m constraints, the method requires a total of $m(2l+1)$ parameters, where l is the number of violation levels defined for each constraint. This large number of parameters makes the quality of the results highly dependent on the chosen values.

In 1995, Le Riche *et al.* proposed the *Segregated Genetic Algorithm (SGGA)* [LER 95] as a different approach for handling the constraints by static penalization. SSGA uses two different penalty functions at the same time. The first penalty function performs weak penalties while the second applies strong penalties. SSGA aims to overcome the problem of too high or too low penalties discussed in section 3.2. It creates from the population two groups of individuals that coexist and cooperate. Each group is evaluated using one of the two penalty coefficients defined as parameters of the algorithm. The two groups are *segregated* during the selection step, where each individual is sorted into a list using one of the two penalties. However, the variation operators are applied on a single one-ranked population that combines both groups. The new population is made by selecting the best solutions in both lists. Two advantages result from this strategy:

1) The search space is explored with two different trajectories, one for each group. Also, thanks to the hybridization of the two groups, the population can avoid local optima.

2) In constrained optimization problems, the global optimum is often on the boundary between the feasible and unfeasible areas. The hybridization between the two groups favors the exploration of the borders and the global optimum is thus localized quite easily.

This algorithm has been tested in [MIC 96b] and has performed better than the static penalty method of Homaifar *et al.* [HOM 94]. However, the quality of the results depend on the choice of the penalty coefficients.

3.2.2. *Dynamic penalties*

With the dynamic strategy, the values of the penalty coefficients are modified along the evolution according to a user-defined schedule - usually increased - in order to ensure the generation of feasible individuals in the end.

Joines and Houck [JOI 94] proposed to evolve the penalties as follows:

$$p(\mathbf{x}) = (C \times t)^{\delta} \sum_{j=1}^{m} v_j^{\beta}(\mathbf{x}),$$

where t is the current generation and $C,\ \delta$, and β are constant values and parameters of the method. A good choice of these parameters reported by Joines and Houck [JOI 94] is $C = 0.5,\ \delta = \beta = 2$.

This method requires much fewer parameters than the method using static penalties and gives better results thanks to the increasing selection pressure on the unfeasible solutions due to the term $(C \times t)^{\delta}$ of the penalty function. However, often, the factor $(C \times t)^{\delta}$ increases very quickly and the pressure becomes too strong. This affects the exploration process and reduces the chances of avoiding possible local optima.

Another approach based on dynamic penalties was proposed by Michalewicz and Attia [MIC 94] for their system GENOCOP II. It is the second version of the system GENOCOP (GEnetic algorithm for Numerical Optimization of COnstrained Problems). The latter had the disadvantage of being able to only handle linear constraints (see section 3.4.2).

The algorithm begins at first by distinguishing linear constraints LC and nonlinear constraints NC. It then builds an initial population which has to satisfy the set LC. The feasibility of the population according to this set is maintained during the evolution thanks to some special operators in the GENOCOP system (see section 3.4.2), which transform a feasible solution into another feasible one.

To handle the nonlinear constraints, Michalewicz and Attia were inspired by the cooling strategy of the Simulated Annealing used to define a penalization function:

$$p(\mathbf{x},\tau) = \frac{1}{2\tau}\sum_{j=1}^{m} {v_j}^2(\mathbf{x}),$$

where τ is the temperature of the system, which is a parameter of the algorithm. τ is decreased in every generation according to a "cooling" scheme defined beforehand. It aims to increase the pressure on unfeasible individuals along the evolution. The algorithm stops when τ reaches the minimal temperature, τ_f, which is also a parameter of the algorithm. Experiments made by the authors [MIC 94] showed that their algorithm may converge in a few iterations with a good choice of the "cooling" scheme, as it may give unsatisfactory results with other schemes.

3.2.3. *Adaptive penalties*

The main idea of the methods based on adaptive penalties is to introduce in the penalization function, a component dependent on the state of the search process in a given generation. Thus, the weight of the penalty is adapted every iteration, and it can be increased or decreased according to the quality of the solutions in the population. Numerous methods have been proposed in this category, of which three are presented in this chapter: the method of Hadj-Alouane and Bean [HAD 97], the method of Smith and Tate [SMI 93] and the method of Ben Hamida and Schoenauer [BEN 00, BEN 02].

Method of Hadj-Alouane and Bean

With this method, the weight of the penalty depends on the quality of the best-found solution at generation t. The penalty function is defined as follows:

$$p(\mathbf{x}) = \alpha(t)\sum_{j=1}^{m} v_j^2(\mathbf{x})$$

where $\alpha(t)$ is updated at each generation t as follows:

$$\alpha(t+1) = \begin{cases} (1/\beta_1).\alpha(t), & \text{if } \mathbf{x}_b \in \mathcal{F} \text{ over the last } k \text{ generations} \\ \beta_2.\alpha(t) & \text{if } \mathbf{x}_b \in (\mathcal{S}-\mathcal{F}) \text{ over the last } k \text{ generations} \\ \alpha(t) & \text{else,} \end{cases}$$

where $\mathbf{x}_b$ is the best solution in the current population and $\beta_1, \beta_2 > 1$ (with $\beta_1 \neq \beta_2$ to avoid the cycles). In other words, this method decreases the value of the component $\alpha(t+1)$ at generation $t+1$ if all the best solutions over the last k generations were feasible and increases its value in the opposite case (all the best solutions were unfeasible). On the other hand, if during these k generations, there was at the same time best feasible solutions and others unfeasible, $\alpha(t+1)$ keeps the same value as $\alpha(t)$.

Hadj-Alouane and Bean aimed to increase the penalties only if they pose a problem for the search process, otherwise they are reduced. However, the strategy of adaptation is based only on the state of the best individual over the k last generations. It does not consider the general state of the population.

Method of Smith and Tate, 1993

The adaptive penalty function proposed by Smith and Tate incorporates, as the previous method, a component indicating the state of evolution of the search process as well as a component indicating the degree of violation of the constraints. The first component depends on the quality of the best solution found so far during the evolution. The second component is determined by the distance of the best unfeasible solutions to the feasible region. The purpose of this function is to expand the search space by introducing interesting unfeasible solutions (close to the feasible domain), which may facilitate the search process when the optimum is located on the boundary of $\mathcal{F}$.

The penalty function is defined as follows:

$$p(\mathbf{x}) = (f_{feas}(t) - f_{all}(t)) \sum_{j=1}^{m} (v_j(\mathbf{x})/q_j(t))^k,$$

where $f_{all}(t)$ is the value of the objective function (without penalty) of the best solution found during the evolution (up to the current generation t), $f_{feas}(t)$ is the fitness of the best feasible solution found over the evolution, $q_j(t)$ is an estimate of the feasibility expansion threshold for each constraint and k is a constant that adjusts the "severity" of the penalty function.

Note that the thresholds $q_j(t)$ are dynamic; they are adjusted during the search process. For example, it is possible to define $q_j(t) = q_j(0)/(1 + \beta_j.T)$, where $q_j(0)$ is the maximum threshold and β_j is a parameter to be set manually.

However, the authors recommend to change the adjustment technique of $q_j(t)$ according to the nature of the problem.

As for the method of Hadj-Alouane and Bean, the penalty function does not consider the general state of the population. Only the performance of the best feasible and unfeasible solutions are considered. Besides, this method has a further difficulty due to the choices to be made in the adjustment of component $q_j(t)$.

Method of Ben Hamida and Schoenauer, ASCHEA

Ben Hamida and Schoenauer [BEN 00] proposed the Adaptive Segregational Constraint Handling Evolutionary Algorithm (ASCHEA). The main idea of ASCHEA is to enhance the exploration around the boundaries of $\mathcal{F}$ by maintaining both feasible and unfeasible individuals in the population. In order to achieve this goal, ASCHEA relies on three main ingredients:

1) *A population-based adaptive penalty function* that uses global information on the population to adjust the penalty coefficients:

$$p(\mathbf{x}) = \alpha(t) \sum_{j=1}^{m} v_j(\mathbf{x}) \qquad [3.5]$$

where $v_j(\mathbf{x})$ $j = 1..m$ are the constraint violation measures (equation 3.3). Increasing the value of the penalty coefficient $\alpha(t)$ in equation 3.5 clearly favors feasible individuals in subsequent selections, while decreasing unfeasible individuals.

Hence, in order to try to maintain a given proportion τ_{target} of feasible individuals, ASCHEA adapts the following strategy:

$$\alpha(t+1)(\mathbf{x}) = \begin{cases} \alpha(t)/fact & \text{if } (\tau_t > \tau_{target}) \\ \alpha(t) * fact & \text{otherwise} \end{cases} \qquad [3.6]$$

where τ_t denotes the proportion of feasible solutions in the current population and $fact > 1$ is a user-defined parameter.

2) *A constraint-driven recombination*, where in some cases feasible individuals can only mate with unfeasible individuals. It is known, in many real-world problems, that the constrained optimum lies on the boundary of the feasible domain (e.g. when minimizing some cost with technological

constraints). In order to both achieve better exploration of the boundary region and attract unfeasible individuals more rapidly towards feasible ones, ASCHEA uses a selection/seduction mechanism, choosing the mate of feasible individuals to be unfeasible. However, to also explore the feasible region, this mechanism is only applied when too few feasible individuals are present in the population. Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be two individuals:

```
if $0 < \tau_t < \tau_{target}$ and $\mathbf{x}_1$ is feasible, then
    select $\mathbf{x}_2$ among unfeasible individuals only
otherwise
    select $\mathbf{x}_2$ according to its fitness only
```

3) *A segregational selection* that distinguishes between feasible and unfeasible individuals. It starts by selecting, without replacement, feasible individuals based on their fitness, until $\tau_{select} * \mu$ individuals have been selected (τ_{select} is a user-defined proportion and μ is the population size), or no more feasible individuals are available. The population is then filled using standard deterministic selection on the remaining individuals, based on the current penalized fitness. So, only a proportion τ_{select} of feasible individuals is considered superior to all unfeasible points.

Constraint satisfaction difficulty differs from one constraint to another one, mainly for active constraints. To adapt the search process to the constraint difficulty and thus ensure a better exploration around the boundaries, Ben Hamida and Schoenauer proposed an improved version of ASCHEA [BEN 02], where each constraint is handled independently. Hence, the adaptive penalty $\alpha(t)$ is extended to several penalty coefficients, $\alpha_j(t)$- one for each constraint j:

$$p(\mathbf{x}) = \sum_{j=1}^{m} \alpha_j(t) v_j(\mathbf{x}),$$

Each coefficient is adapted, according to a single penalty (equation 3.6), according to $\tau_t(j)$, which is the proportion of individuals satisfying the constraint. The idea is to have individuals on both sides (feasible and unfeasible) of the corresponding boundary.

Another component has been introduced in the improved version of ASCHEA [BEN 02] to allow it to better handle the equality constraints. This component transforms the qualities $h_j(\mathbf{x}) = 0$ into two inequality constraints, $-\epsilon_j(t) \leq h_j(\mathbf{x}) \leq \epsilon_j(t)$, where the value of ϵ_j is adjusted along the evolution. The objective is to start with a large feasible domain to enhance the search space exploration process, and to reduce it progressively in order to bring it as close as possible to the real domain with a null measure at the end of the evolution, as illustrated in Figure 3.5. At each generation t, unfeasible solutions are then *pushed* to the new feasible space $\mathcal{F}_{h_j}(t+1)$, thanks to the penalization and the selection/seduction strategies of ASCHEA.

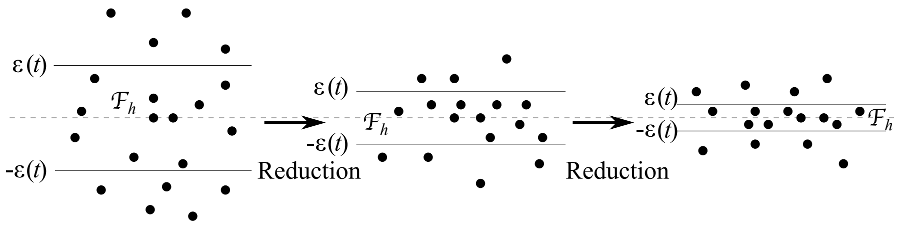


Figure 3.5. *Progressive reduction of the feasible domain $\mathcal{F}_{h_j}$ along the evolution corresponding to the equality constraint h_j using adaptive adjustment of ϵ_j at each generation t*

At the end of evolution, $\epsilon_j(t)$ takes very small values close to 0, which numerically satisfies the equality constraint.

Thanks to its components, ASCHEA has a high capacity for exploring and exploiting the best feasible and unfeasible solutions, especially at the end of the evolution. It has given good results for several benchmarks [BEN 00, BEN 02] but remains costly in terms of the number of evaluations.

3.2.4. *Self-adaptive penalties*

The main characteristic of the methods based on the self-adaptive penalties is that they do not require additional parameters. In fact, the penalties are adapted solely on the basis of the information extracted from the population. Two classes of methods have been proposed in this category: those that consider the state of the population for several generations such as the method of Coello [COE 99] which is the first technique published in this category, and those that consider only the state of the population in the current generation, such as the method of Farmani and Wright [FAR 03] and that of Tessema and Yen [TES 06]. Below, we present a method in each class.

Method of Coello, 1999

It is based on the principle of co-evolution, where a population of penalties co-evolves with the population of solutions. The population of solutions $P1$ is evaluated according to the following formula:

$$eval(\mathbf{x}) = f(\mathbf{x}) - (\alpha_1 \times \sum_{j=1}^{m} v_j(\mathbf{x})) + \alpha_2 \times \theta(\mathbf{x}))$$

where $\sum_{j=1}^{m} v_j(\mathbf{x})$ is the sum of the measures of the constraint violations by the solution $\mathbf{x}$, $\theta(\mathbf{x})$ is the number of constraints violated by $\mathbf{x}$ and α_1 and α_2 are two penalty coefficients.

A population $P2$ of vectors of penalty coefficients $A_j = (\alpha_1, \alpha_2)$ is maintained and co-evolves in parallel with the population of solutions $P1$. Each vector A_j of $P2$ is used to evaluate all individuals of $P1$ for a given number of generations, after which the fitness of the corresponding vector A_j is computed by using the following formula:

$$\varphi(A_j) = \sum_{i=1}^{N} \frac{eval(\mathbf{x}_i)}{N_f} + N_f$$

where $\varphi(A_j)$ is the average fitness of A_j and N and N_f are, respectively, the size of the population $P1$ and the number of its feasible solutions. Thanks to the added quantity N_f, the GA is encouraged to move towards the region containing a large number of feasible solutions. The best performance corresponds to the vectors A_j that make it possible to generate more feasible solutions which are closer to the optimum.

The genetic operators are applied to the population $P2$ after the evaluation of all vectors A_j. Thus, the penalty coefficients are adjusted automatically according to the information provided by the evolution of the population of solutions $P1$.

The major disadvantage of this method is its cost due to the large number of evaluations.

Method of Tessema and Yen, 2006

The SAPF method (*Self-Adaptive Penalty Function*) of Tessema and Yen [TES 06] is based on the distribution of the current population in the search space to adjust the penalties. The SAPF method is summarized in the following four steps:

1) Normalize the values of $f(\mathbf{x})$ for all solutions in the population according to the following formula:

$$\widetilde{f}(\mathbf{x}) = \frac{f(\mathbf{x}) - \min_{\mathbf{x}} f(\mathbf{x})}{\max_{\mathbf{x}} f(\mathbf{x}) - \min_{\mathbf{x}} f(\mathbf{x})}$$

where $\min_{\mathbf{x}} f(\mathbf{x})$ and $\max_{\mathbf{x}} f(\mathbf{x})$ correspond to the fitness without penalty of, respectively, the best and worst solutions in the population;

2) Normalize the measures $\widetilde{v}$ of the constraint violation $v(\mathbf{x})$ for all the solutions in such a way that $\widetilde{v}(\mathbf{x}) \in [0, 1]$;;

3) For each solution $\mathbf{x}$, compute the distance, $d(\mathbf{x})$, as follows:

$$d(\mathbf{x}) = \begin{cases} \widetilde{v}(\mathbf{x}) & \text{if all solutions are unfeasible} \\ \sqrt{\widetilde{f}(\mathbf{x})^2 + \widetilde{v}(\mathbf{x})^2} & \text{otherwise} \end{cases}$$

4) Evaluate the solutions using the following formula:

$$eval(\mathbf{x}) = d(\mathbf{x}) + (1 - r_t)\alpha_1(\mathbf{x}) + r_t\alpha_2(\mathbf{x})$$

where r_t is the proportion of feasibility of the population at generation t, defined by the ratio of the number of feasible solutions to the size of the population, and α_1 and α_2 are two penalty functions defined as follows:

$$\alpha_1(\mathbf{x}) = \begin{cases} 0 & \text{if } r_t = 0 \\ \widetilde{v}(\mathbf{x}) & \text{otherwise} \end{cases} \quad \alpha_2(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \text{ is feasible} \\ \widetilde{f}(\mathbf{x}) & \text{otherwise} \end{cases}$$

The steps described above allow the SAPF method to automatically adapt the penalties according to the distribution of the population in the search space while taking into account:

– the proportion of feasible solutions;

– the values of the objective function $f(\mathbf{x})$;

– the distances of unfeasible solutions to the feasible space $\mathcal{F}$.

This technique has given good results for several test cases [TES 06]. However, it has a low performance when the feasible region is too small or has a null measure, since the algorithm focuses more on the search for feasible solutions than on the optimization of the objective function.

3.2.5. *Stochastic ranking*

Proposed by Runarsson and Yao in 2000 [RUN 00], this method creates a balance between the objective function and the penalty function, based on a stochastic ranking of the individuals, as described below.

If we consider that the solutions are evaluated with equation 3.1, then the penalty function is defined as follows:

$$p(\mathbf{x}) = r_t.\sum_{j=1}^{m} v_j(\mathbf{x}) = r_t\theta(\mathbf{x}),$$

where r_t is the penalty coefficient and $\theta(\mathbf{x})$ is the violation measure.

Finding a good strategy to adjust r_t in each generation, while avoiding *over-penalization* and *under-penalization*, is by itself an optimization problem.

To overcome this difficulty, Runarsson and Yao propose the Stochastic ranking. They define a probability P_f to decide whether to use the objective function or to use the penalty function for the comparison. Thus, two adjacent individuals $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$, of which at least one is unfeasible, have a probability P_f to be compared according to the values of their objective function, otherwise they are compared according to their constraint violation measures. If both individuals are feasible, $P_f = 1$. The sorting algorithm of the Stochastic Ranking method is shown in Algorithm 1:

```
for i=1 To N do
    change ← false
    for j=1 To N–1 do
        $u \leftarrow \mathcal{U}[0,1]$ // $\mathcal{U}[0,1]$ : uniform random number in $[0,1]$
        if ($\theta(\mathbf{x}_j) = \theta(\mathbf{x}_{j+1})$) or ($u < P_f$) then
            if ($f(\mathbf{x}_j) > (f(\mathbf{x}_{j+1})$) then
                Swap($\mathbf{x}_j, \mathbf{x}_{j+1}$)
                change ← true
            end
        end
        else
            if ($\theta(\mathbf{x}_j) > \theta(\mathbf{x}_{j+1})$) then
                Swap($\mathbf{x}_j, \mathbf{x}_{j+1}$)
                change ← true
            end
        end
    end
    if not change then
        break
    end
end
```

Algorithm 1: Sorting algorithm of the Stochastic Ranking method

Stochastic Ranking has been further developed in [RUN 06], using the surrogate models for fitness approximations in order to reduce the total number of function evaluations needed during search. The authors found that the improved version provided the most competitive performance in a set of benchmark problems.

The simplicity of Stochastic Ranking has made it suitable for use in different domains. It provides robust results for different problems and benchmarks [LEG 07].

3.3. Superiority of feasible solutions

The superiority of the feasible individuals approach is based on the following heuristic rule: “any feasible solution is better than any unfeasible solution". This property is not guaranteed in the case of penalty methods discussed above.

The main purpose of the methods in this category is to give an advantage to the feasible solutions in the selection process. Some additional rules are then introduced in the selection procedure. Some techniques use, besides the feasibility rules, a special penalty function.

3.3.1. *Special penalization*

Some of the methods using the penalization method have as main objective the implementation of the superiority of the feasible solution. In this way, we consider that they could be classified in this category rather than another. The first method in this sub-class is that of Powell and Skolnick published in 1993 [POW 93] described below.

Method of Powell and Skolnick

The method of Powell and Skolnick [POW 93] also uses the penalty functions, but in a different way. The purpose is to map the fitness of all unfeasible solutions in the interval $(1, +\infty)$ and the fitness of all feasible solutions in the interval $[-\infty, 1)$ (for a minimization problem). Indeed, to evaluate unfeasible solutions, in addition to the constraint violation measures, it uses the evaluation of feasible solutions as follows:

$$p(\mathbf{x}) = r \sum_{j=1}^{m} v_j(\mathbf{x}) + \theta(t, \mathbf{x})$$

where r is a constant parameter of the algorithm.

The component $\theta(t, \mathbf{x})$ is a function of the population state at the current generation t, and it has a great influence on the assessment of unfeasible individuals:

$$\theta(t, \mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \in \mathcal{F} \\ \max\{0, \delta\}, & \text{otherwise.} \end{cases}$$

with

$$\delta = \max_{\mathbf{y} \in \mathcal{F}} \{f(\mathbf{y})\} - \min_{\mathbf{y} \in (\mathcal{S} - \mathcal{F})} \left\{ f(\mathbf{y}) + r \sum_{j=1}^{m} v_j(\mathbf{y}) \right\}$$

With this additional heuristic, the performance of the unfeasible solutions depends on that of the feasible solutions: the best fitness of an unfeasible solution could not be better than the worst fitness of any feasible solution ($\max_{\mathbf{x}\in\mathcal{F}}\{f(\mathbf{x})\}$).

This method requires choosing a single parameter, r. The use of a small value allows the algorithm to explore the unfeasible region in parallel with the feasible domain, but if r is large, few unfeasible individuals survive in the population. Otherwise, the success of the method also depends on the topology of the feasible search space. Experiments published in [MIC 95] indicate that for problems where $\mathcal{F}$ is too small, the method may fail.

3.3.2. *Feasibility rules*

The feasibility rules were introduced by Deb in 2000 [DEB 00]. Deb proposed to handle constraints using three basic rules to rank individuals using pair-wise comparisons. This approach is very simple to implement and does not need user-defined parameters. For these reasons, it was adopted by a large number of recent constraint-handling methods. Some of them are presented in the third paragraph of this section. Moreover, this approach was further developed by Takahama and Sakai in their ε-constrained method [TAK 04]. A detailed representation is given in the second subsection.

Deb's method

The method proposed by Deb [DEB 00] avoids the computation of the objective function in the unfeasible region. The proposed approach is based on the idea that any individual in a constrained search must first comply with the constraints and then with the objective function. It uses a binary tournament selection, where two solutions are compared according to the following criteria:

1) Every feasible solution is better than every unfeasible solution.

2) Among two feasible solutions, the one with the best fitness is selected.

3) Among two unfeasible solutions, the one having the smallest violation measure is selected.

The evaluation of unfeasible solutions does not use a penalty coefficient but is based on the the constraint violation measures and the fitness of the feasible solutions:

$$eval(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{x} \in \mathcal{F}, \\ f_{max} + \sum_{j=1}^{m} v_j(\mathbf{x}), & \text{otherwise,} \end{cases}$$

where f_{max} is the value of the objective function of the worst feasible solution in the population. To maintain the diversity in the feasible domain, the method uses a niching technique applied during the selection step [DEB 89].

This method has the advantage of not requiring additional parameters. In addition, it does not compute the objective function in the unfeasible region. However, as for the method of Powell and Skolnick, it may fail if the ratio $|\mathcal{F}|/|\mathcal{S}|$ is too small.

ε-Constrained method

Takahama and Sakai [TAK 04] proposed the ε-constrained method, (called the α-constrained method in an earlier version), which samples the feasible space. It transforms the constrained problem to an unconstrained one using ε-level comparison. ε is a satisfaction level of constraints used to express how much a search point satisfies the constraints. This heuristic could be considered as a relaxation of the limit of the feasible space based on the sum of constraint violations.

The value of $\varepsilon > 0$ determines the so-called ε-level comparisons between a pair of solutions $\mathbf{x_1}$ and $\mathbf{x_2}$ with objective function values $f(\mathbf{x_1})$ and $f(\mathbf{x_2})$ and the sum of constraint violations $V(\mathbf{x_1})$ and $V(\mathbf{x_2})$ computed according to equation 3.4. If $\mathbf{x_1}$ and $\mathbf{x_2}$ are feasible according to the ε-level or if they have the same sum of constraint violations, they are compared using their objective function values. If both solutions are unfeasible, they are compared based on their sum of constraint violations as follows:

$$(f(\mathbf{x}_1), V(\mathbf{x_1})) <_{\varepsilon} (f(\mathbf{x}_2), V(\mathbf{x_2})) \Leftrightarrow \begin{cases} f(\mathbf{x}_1) < (f(\mathbf{x}_2) \text{ if } V(\mathbf{x_1}), V(\mathbf{x_2}) \leq \\ f(\mathbf{x}_1) < (f(\mathbf{x}_2) \text{ if } V(\mathbf{x_1}) = V(\mathbf{x_2}) \\ V(\mathbf{x}_1) < V(\mathbf{x_2}) \text{ otherwise} \end{cases}$$

$$(f(\mathbf{x}_1), V(\mathbf{x_1})) \leq_{\varepsilon} (f(\mathbf{x}_2), V(\mathbf{x_2})) \Leftrightarrow \begin{cases} f(\mathbf{x}_1) \leq (f(\mathbf{x}_2) \text{ if } V(\mathbf{x_1}), V(\mathbf{x_2}) \leq \\ f(\mathbf{x}_1) \leq (f(\mathbf{x}_2) \text{ if } V(\mathbf{x_1}) = V(\mathbf{x_2}) \\ V(\mathbf{x_1}) < V(\mathbf{x_2}) \text{ otherwise} \end{cases}$$

This is a lexicographical ordering mechanism in which the minimization of the sum of constraint violations precedes the minimization of the objective function of a given problem. The initial ε-level $\varepsilon(0)$ is the constraint violation of the best unfeasible solution, i.e. the top $V(\mathbf{x})$ in the initial search points. The ε-level is updated using the following equation until the number of iterations t reaches the control generation T_c which is a user-defined parameter. Then, the ε-level is set to 0 in order to obtain a final population with minimum constraint violations:

$$\varepsilon(t) = \begin{cases} \varepsilon(0)(1 - \frac{t}{T_c}) & 0 < t < T_c \\ 0 & t > T_c \end{cases}$$

A successful attempt to find a more efficient search algorithm for the ε-constrained method was reported in [TAK 06], where a DE variant (DE/rand/1/exp) (section 2.5) and a gradient-based mutation operator (acting as a local search engine) were employed. This version obtained the best overall results in a competition on constrained real-parameter optimization in 2006, in which a set of 24 test problems were solved [TAK 06].

Methods based on the feasibility rules

Thanks to their simplicity, the feasibility rules are widely used for CEO. However, as described in the original Deb's method [DEB 00], this strategy needs extensions to enhance the population diversity and then avoid the premature convergence. Deb proposed to use a niching technique. Other authors proposed to use parallel population or hybrid approaches. The following paragraphs give a short description of these methods.

In 2003, Zhang and Xie introduced the DEPSO method: Hybrid Particle Swarm with Differential Evolution Operator [ZHA 03]. DEPSO combines the PSO (section 2.7) with Differential Evolution (DE) operators (section 2.5). DE operators are used to maximize diversity in the population. DEPSO also introduces the bell-shaped mutation [ZHA 03] used to mutate the best previous position recorded. DEPSO was tested on 11 test functions [ZHA 03]. All constraints were handled by the feasibility rules. The DEPSO seems to perform well for the problems with integer variables with the help of the bell-shaped mutations. However, it seems not very efficient for handling problems with an extremely small feasible space.

He and Wang [HE 07] proposed a method based on the same idea as DEPSO, but it combines PSO and a Simulated Annealing (SA) method, used to avoid premature convergence. They called their method HPSO (Hybrid Particle Swarm Optimization). SA is applied to the best solutions of the swarm to help the algorithm escape from local optima [HE 07].

Another approach proposed to use the feasibility rules with a parallel population implementation. One known technique in this class is the Dual Population Genetic Algorithm (DPGA) introduced by Park and Ruy in 2006 [PAR 06]. DPGA evolves, in parallel to the main population, a Preserver, which is a reserve population that provides diversity in the main population. Such implementation was experimented for CEO by Umbarkar *et al.* in their OpenMP-DPGA method [UMB 15], which is a parallel implementation of DPGA using multi-threading. It starts with two randomly generated populations (main population and reserve population). The evolution of each population is obtained by inbreeding between parents from the same population, cross-breeding between parents from different populations and survival selection among the obtained offspring. Experiments on a set of constrained problems showed an increase in the speed up of the algorithm. However, the method could fail to find the global optimum and it is unable to deal with equality constraints.

3.4. Evolving on the feasible region

The methods in this category help the EA to evolve solutions on the feasible region, either by including special techniques to search feasible solutions or by using special operators to maintain the feasibility of the population.

3.4.1. *Searching for feasible solutions*

The methods in this class bring individuals to the feasible space $\mathcal{F}$. It is divided into two sub-classes: repairing unfeasible individuals and sampling the feasible space. A method is presented below for each sub-class.

Repairing methods: GENOCOP III

This is the third version of the GENOCOP System proposed by Michalewicz and Nazhiyath in 1995 [MIC 95]. It is based on the idea of repairing unfeasible solutions to make them feasible, and it also uses some co-evolutions concepts. This method incorporates the original GENOCOP system (described in section 3.4.2) and extends it with an additional module that co-evolves two separate populations. The first population P_s includes points that satisfy the linear constraints of the problem, and they are called the search points. The feasibility of the points within P_s (relative to linear constraints) is maintained thanks to the special operators of the GENOCOP system (see section 3.4.2). The second population P_r includes points that satisfy all constraints of the problem (linear and nonlinear), and they are called reference points. The reference points $\mathbf{r}_i$ of P_r, being feasible, are evaluated directly with the objective function ($eval(\mathbf{r}) = f(\mathbf{r})$). However, the search points P_s, which are not feasible, are repaired before they are evaluated.

A search point $\mathbf{s}$ is replaced by a point $\mathbf{z}$ in population P_s with a replacement probability p_r. It is also noted that there is an asymmetry between the evolution of the two populations P_s and P_r. Indeed, the application of reproduction operator and of the selection procedure to the population P_s is done at each generation, whereas it is only done in all k generations for the population P_r, where k is a parameter of the method.

The co-evolution strategy of the two populations is given by the main procedure of the system GENOCOP III presented below.

```
$t \leftarrow 0$
initialize $P_s(t)$, $P_r(t)$
evaluate $P_s(t)$, $P_r(t)$
**repeat**
    $t \leftarrow t + 1$
    select $P_s(t)$ from $P_s(t-1)$
    reproduction of $P_s(t)$
    evaluate $P_s(t)$
    **if** $t \bmod k = 0$ **then**
        reproduction of $P_r(t)$
        select $P_r(t)$ from $P_r(t-1)$
        evaluate $P_r(t)$
    **end**
**until** *Stopping criterion satisfied*
```

Algorithm 2: The GENOCOP III algorithm

Note that the reproduction is accomplished before the selection in the evolution process of P_r, due to the low probability of generating a feasible offspring. Thus, the offspring are created first, then the best feasible individuals among the parents and the offspring are selected to form the new population.

The advantages of GENOCOP III are that it does not evaluate the objective function in the unfeasible space and that it always returns a feasible solution. By contrast, the algorithm struggles to create the population of reference points if the ratio $|\mathcal{F}|/|\mathcal{S}|$ is very small. In particular, if the feasible region is not connected and if the population P_r has been initialized in a single component of $\mathcal{F}$, then the system will encounter difficulties to generate new feasible individuals in the other components of $\mathcal{F}$.

Sampling the feasible space

The first method which aimed to sample the feasible space was proposed by Schoenauer and Xanthakis in 1993 [SCH 93]. It is based on the concept of behavioral memory of the population: *"the population contains not only information on the strict optimum, but also information on its behavior in the past"*.

It creates a feasible population by processing the various constraints of the problem one by one and in a particular order. The algorithm begins with a random population. Then, for each constraint, it evolves the individuals until a certain percentage of the population becomes feasible for the constraint being considered, while continuing to respect the previous constraints. There are $q+1$ steps for q constraints to satisfy. The population obtained at the end of each step is used as a starting point for the evolution of the next constraint. A linear order for processing constraints must be defined. For the q first steps, the fitness in step i is a function $M(g_i(x))$ that is maximal when the constraint $g_i(x) \leq 0$ is satisfied. Individuals who do not satisfy the constraints g_1 to g_{i-1} are eliminated from the population by assigning them a zero fitness.

The objective function is optimized in the last step with the death penalty method for unfeasible points. In this step, the population may be located in a very small area in the search space owing to the sequential process of the constraints. This problem can be solved using a niching method to maintain diversity in each step.

This approach has the advantage of avoiding the evaluation of the objective function in the unfeasible region; however, it can fail if the feasible domain is very small or disjointed. In addition, the sampling procedure requires choosing a linear order for the treatment problem constraints. This choice greatly influences the quality of the results

3.4.2. *Maintaining feasibility using special operators*

All the methods of this category maintain the feasibility of the population. They use specific reproduction operators to generate feasible offspring from feasible parents (closed operators on $\mathcal{F}$).

GENOCOP system

The first version of GENOCOP (GE*netic algorithm for* N*umerical* O*ptimization of* CO*nstrained* P*roblems*) was proposed in 1991 by Michalewicz and Janikow [MIC 91].

The system deals only with the problems subject to linear constraints. It begins by eliminating the equality constraints by eliminating a number of variables of the problem, which are replaced by linear combinations of the remaining variables. The inequality constraints are then reformulated by replacing the variables eliminated by their linear combinations. The remaining constraints, being linear, form a convex feasible space. Thus, it is quite easy to define closed operators that maintain the feasibility of the solutions.

For example, the arithmetic crossover of two feasible points $\mathbf{x}$ and $\mathbf{y}$ produces an offspring $\mathbf{z} = a\mathbf{x} + (1 - a)\mathbf{y}$, where $a = \mathcal{U}[0, 1]$ is a random number drawn uniformly in $[0.1]$. It is then guaranteed that $\mathbf{z}$, in a convex domain, is always feasible.

Another crossover operator, called heuristic crossover, has also been added to GENOCOP. This operator generates a child $\mathbf{z}$ from parents $\mathbf{x}$ and $\mathbf{y}$ selected such that the fitness $f(\mathbf{y})$ is better than $f(\mathbf{x})$ by applying the following rule:

$$\mathbf{z} = r.(\mathbf{y} - \mathbf{x}) + \mathbf{y}, \text{ where } r = \mathcal{U}[0, 1].$$

In this way, a further exploration is performed in the neighborhood of the promising solution $\mathbf{y}$ outside the segment $(\mathbf{y} - \mathbf{x})$. If an unfeasible solution is generated, then another random value r is selected and another offspring is created. If no new feasible solution is found after a given number of attempts, the operator gives up and produces no offspring.

For the mutation, GENOCOP proceeds in two steps. It first determines the current domain, $dom(x_i)$, for each component x_i of a solution vector x, which is a function of linear constraints and remaining values of the solution vector x. The new value of x_i is taken from this domain.

This method has given good results for problems with convex feasible space. However, it can be relatively expensive, as some crossover operators such as the heuristic crossover may require several iterations before generating a feasible offspring.

Searching on the feasible region boundary

In many cases, for constrained optimization, some constraints are active at the optimum. Thus, the optimum is located on the boundary of the feasible space, $\mathcal{F}$. Michalewicz and Schoenauer proposed an original approach that favors an effective exploration of the boundary of the feasible region [SCH 96a, SCH 97, SCH 98]. They introduced an evolutionary algorithm that starts from an initial population of points randomly selected on the boundary of $\mathcal{F}$. These solutions are then evolved while keeping their feasibility thanks to a set of "closed" genetic operators on $\mathcal{F}$.

The boundary is assumed to be a regular surface $\mathbf{S}$ of dimension $n-1$ in the space $\mathbb{R}^n$. The used operators must be able to generate any point of the surface $\mathbf{S}$ (Figure 3.6) and must respect the following conditions:

1) The crossover must be able to build the points in the neighborhood of both parents.

2) The mutation must be ergodic and must respect the principle of strong causality: a small change in the solution should cause a small change in the corresponding fitness.

Schoenauer and Michalewicz proposed several closed operators whose application depends on the type of surface of the boundary $\mathcal{F}$, such as a specialized crossover and mutation operators for spherical and hyperboloidal surfaces [SCH 96a, SCH 97].

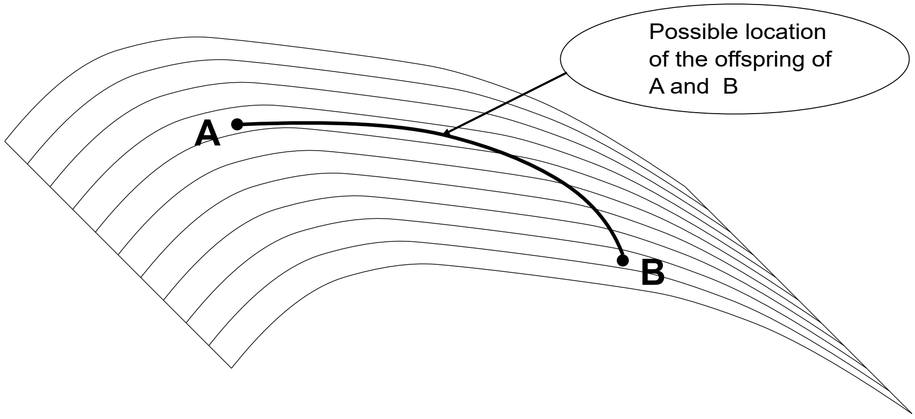


Figure 3.6. Crossover operator on the surface

We can cite as an example the *curve-based operators*: from a curve joining two different points on the surface, a crossover operator can be defined by choosing one or two offspring on that curve. We can also mention the operators based on the geodesic curve or the plane operators based on the curves resulting from the intersection of the surface S with two-dimensional planes.

This class of method has the advantage that it does not need to deal with unfeasible solutions, but it has the great disadvantage of only solving problems whose optimum is on the boundary of the feasible region. In addition, many difficulties may be encountered in the design of the genetic operators that are specific to the problem to solve.

Homomorphous mapping

Proposed in 1999 by Koziel and Michalewicz [KOZ 99b], this method uses a set of decoders to transform a constrained problem into an unconstrained one. It evolves a population of encoded individuals, where each of them corresponds to a solution in the real search space. The following conditions must be satisfied to handle constraints with decoders:

1) For each solution $s \in \mathcal{F}$, there exists an encoded solution d;

2) Each encoded solution d corresponds to a feasible solution s;

3) All solutions in $\mathcal{F}$ must be represented by the same number of codes;

4) The encoding/decoding procedure T must not be too complex and should be fast in computing time;

5) A small change in the encoded solution should generate a small change in the corresponding real solution.

The "homomorphous mapping" is a technique of encoding/decoding any feasible search space $\mathcal{F}$ and an n-dimensional cube defined as $[-1,1]^n$ (Figure 3.7). The encoded solution $\mathbf{y}_0 \in [-1,1]^n$ of a point $\mathbf{x}_0 \in \mathcal{F}$ is obtained by a projection between the half-segment defined by the point $\mathbf{y}_0$ and the center of the cube $\mathbf{O}$ and the half-segment defined by the point $\mathbf{x}_0$ and the reference point $\mathbf{r}_0 \in \mathcal{F}$. Thus, the encoded point $\mathbf{y}_0 \in \mathcal{F}$ corresponding to $\mathbf{x}_0$ is defined as: $\mathbf{y}_0 = (\mathbf{x}_0 - \mathbf{r}_0).\tau$, where $\tau = \frac{||\mathbf{y}_M||}{||\mathbf{x}_M - \mathbf{r}_0||}$. $\mathbf{y}_M$ is determined with a dichotomous search procedure.

This technique can only be applied for convex feasible spaces $\mathcal{F}$, but a generalization has been proposed in the case of non-convex space by introducing an additional encoding/decoding step. However, this generalization of the encoding technique may not respect the fifth condition for the validity of a decoder on the strong causality. The applicability of the method is therefore very limited.

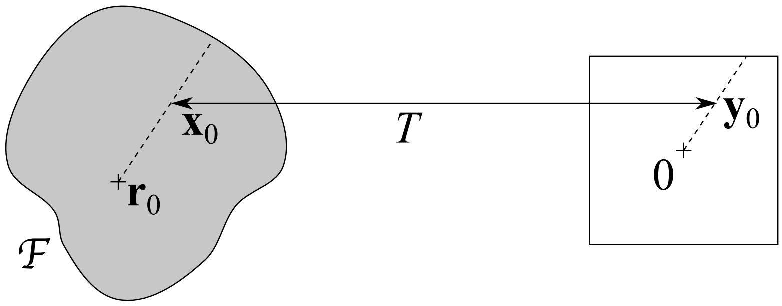


Figure 3.7. *Example of a projection of points between the cube* $[-1.1]^n$ *and the feasible space* $\mathcal{F}$ *(two-dimensional case)*

3.5. Multi-objective methods

The multi-objective approach relies on the idea of transforming the given constraints into additional objective functions to be minimized, the violation measure of each constraint v_j $(j = 1, \cdots, m)$ can be handled as a separate objective in addition to the main objective function f. The first multi-objective method in the category was introduced by Parmee and Purchase in 1994 [PAR 94] for the optimization of a gas turbine design with a heavily constrained search space. They used the multi-objective method proposed by Schaffer [SCH 85], called "Vector Evaluated Genetic Algorithm" (VEGA), which aims to find feasible points to create a set of regions of $\mathcal{F}$ for local search. The objective function is then optimized separately by a genetic algorithm using some special operators in order to help the algorithm remain in the feasible region. This method was complex to implement, but the idea has inspired several researchers and has given birth to a wide range of multi-objective methods for handling constraints.

A common approach to handling constraints with multi-objective techniques is to consider the sum of the constraint violations as a second objective. Hence, it becomes a bi-objective optimization problem. A second approach consists of transforming the constrained problem into an unconstrained multi-objective problem, where the original objective function and each constraint are treated as separate objectives. The methods of the second approach can also be classified into two sub-categories: (1) those that use non-Pareto concepts (mainly based on multiple populations) and (2) those that use Pareto concepts (ranking and dominance) as the selection criteria [COE 02b].

3.5.1. *Bi-objective techniques*

This approach transforms a constrained problem into an unconstrained one as follows:

Minimize $F(X) = f(X), G(X)$ where $G(X) = \sum^{m+p} max(0, g_i(X))$ where each $g_i(X), i = 1 \cdots, m+p)$ must be normalized (Figure 3.8).

A large number of methods based on this approach were proposed, such as the method of Surry *et al.* [SUR 95], the method of Camponogara and Talukdar. [CAM 97], the method of Ray *et al.* [RAY 00] and the IDEA algorithm of Singh *et al.* [SIN 08]

Method of Surry et al.

The method proposed by Surry *et al.* in 1995 [SUR 95], called COMOGA (*Constrained Optimization by Multi-Objective Genetic Algorithms*), handles constraints as the criteria of a multi-objective problem and optimizes simultaneously the objective function as an unconstrained optimization problem. To do this, all members of the search space S are labeled with some measure of their Pareto ranking R based on constraint violations v_j (counting the number of individuals dominated by each solution). Then, each solution is evaluated by both the Pareto rank and the objective function value f:

$$I_R(\mathbf{x}) = (R_{(v_1,\cdots,v_m)}(\mathbf{x}), f(\mathbf{x})).$$

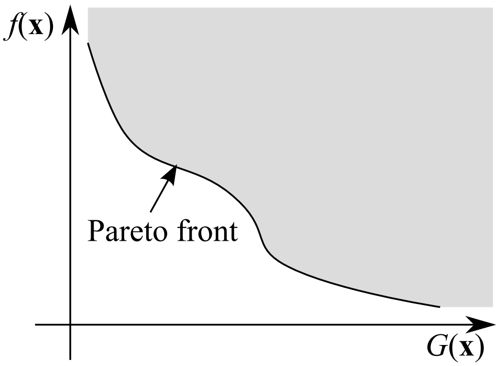


Figure 3.8. Bi-objective approach for CEO

The Pareto ranking is defined using the same sorting technique as in MOGA proposed by Fonseca and Fleming in 1993 [FON 95].

The environmental selection for the next generation proceeds in two steps: at first, $p_{cost} \times N$ individuals are selected using a binary tournament selection based on fitness f. Then, the rest of the individuals $((1 - p_{cost}) \times N)$ are linearly selected according to their ranks R. To avoid the convergence to an unfeasible solution, the value of p_{cost} is adapted dynamically according to the proportion of unfeasible solutions in the population, compared to a reference rate τ. The scheme of this method is summarized in Algorithm 3.

The method was successfully applied to design a gas network (provision and pipe type) [SUR 95] and it gave good results. However, it did not give the same degree of accuracy for other benchmark problems.

```
Compute the constraint violation measures $v_j$ for all solutions;
Compute Pareto ranks $R$ for all solutions using the violation measures $v_j$;
Compute the fitness values $f$;
Select a proportion $p_{cost}$ of solutions using $f$, and the rest proportionally to $R$;
Apply the crossover and mutation operator;
Adjust $p_{cost}$:
**if** *(proportion of feasible individuals) $< \tau$)* **then**
    decrease $p_{cost}$: $p_{cost} \leftarrow (1-\epsilon)p_{cost}$
**end**
***otherwise***
    increase $p_{cost}$: $p_{cost} \leftarrow 1-(1-p_{cost})(1-\epsilon)$, where $0 < \epsilon << 1$.
**end**
```

Algorithm 3: The algorithm of a COMOGA iteration

Method of Camponogara and Talukdar

Camponogara and Talukdar [CAM 97] suggested to handle the constrained optimization problem as a two-objective optimization problem: the first one is the objective function f of the initial problem and the second objective is an aggregation of constraint violations:

$$\Theta(\mathbf{x}) = \sum_{j=1}^{m}(v_j(\mathbf{x}))\ ;\text{ where } v_j(\mathbf{x}) \text{ is given by expression 3.3.}$$

Once the problem is thus redefined, a list of L sets of solutions $\mathbf{S}_i$ with $i \in \{1, ..., L\}$ is built such that:

– $\mathbf{S}_1$ contains only non-dominated solutions in the population;

– $\mathbf{S}_i$, for $i > 1$, contains only non-dominated solutions when sets $\mathbf{S}_j$ with $j \in \{1, ..., i-1\}$ are removed from the population.

Then, pairs of solutions $\mathbf{x}_i \in \mathbf{S}_i$ and $\mathbf{x}_j \in \mathbf{S}_j$ are selected such that $\mathbf{x}_i$ dominates $\mathbf{x}_j$ with $i < j$. These solutions define a search direction as follows:

$$\mathbf{d} = \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}$$

A line search is then applied in the direction of research defined by $\mathbf{d}$ in order to create a better solution $\mathbf{y}$, which dominates $\mathbf{x}_i$ and $\mathbf{x}_j$. The authors proposed to use a uniform search in the segment defined by $\mathbf{x}_j$ and $\mathbf{x}_j + 2(\mathbf{x}_i - \mathbf{x}_j)$ [CAM 97].

This technique is simple to implement; however, it encounters some difficulties in preserving population diversity. Additionally, the use of line search within a GA adds some extra computational cost.

IDEA method of Singh et al.

Singh *et al.* proposed in 2008 the method IDEA, *Infeasibility Driven Evolutionary Algorithm* [SIN 08]. It not only considers the constraints as objectives but also maintains in the population the best unfeasible solutions to try to approach the optimum from both feasible/unfeasible sides of the search space. IDEA then transforms the constrained problem into a bi-objective optimization problem as follows:

$$\text{Minimize} \begin{cases} f(\mathbf{x}) \\ \Theta(\mathbf{x}) = \sum_{j=1}^{m} (R_j(\mathbf{x})) \end{cases}$$

IDEA assigns to each solution $\mathbf{x}$ in the population m ranks $R_j(\mathbf{x})$, corresponding to the m constraints of the problem, based on the violation measures v_j. For a constraint j, the rank 0 corresponds to the solutions respecting this constraint, the rank 1 to the solution having the minimum violation measure and the remaining solutions will have ascending ranks according to the violation measures. $\Theta(\mathbf{x})$ is then the sum of ranks R_j assigned to the solution $\mathbf{x}$. The Pareto rank is then used by the genetic operators in the same way as in the NSGA-II method.

At the replacement step, a proportion λ of the new population is selected from the the set of solutions with $\Theta(\mathbf{x}) > 1$. The goal is to keep along the evolution the best unfeasible solutions.

Thanks to this additional component, IDEA has a better convergence ability than the other methods in the same category. The method showed a high performance and a fast convergence when applied by Singh *et al.* [SIN 09] on a dynamic optimization problem.

3.5.2. *Multi-objective techniques*

In this second category of multi-objective approach, the constrained problem is transformed into an unconstrained one as follows: optimize $F(x) = f(x), f_1(x), f_2(x), \cdots, f_m(x)$, where $f_1(x), f_2(x), \cdots, f_m(x)$ are the constraints of the given problem.

Two schemes are possible within multi-objective approaches: those using the Pareto front and those using other concepts such as multiple populations. In the second sub-class, we cite essentially the method of Coello [COE 02b] presented below. In the first sub-class, there are much more methods. We describe below some of them: the method of Coello and Mezura [COE 02c], the method of Ray *et al.* [RAY 00], the method of Aguirre *et al.* [AGU 04] and the method of Cai and Wang [CAI 06].

Method of Coello

Coello proposed a sub-population-based muli-objective technique using VEGA [COE 02b]. Each constraint is considered as a separate objective for one of the m sub-populations, where m is the number of constraints. The objective function is also optimized by a sub-population. During the selection step, a proportion of the population is selected using the objective function, and the remaining individuals are selected according to their fitness based on the corresponding constraint. This fitness is computed as follows:

$$eval(\mathbf{x}) = \begin{cases} g_j(\mathbf{x}), & \text{if } g_j(\mathbf{x}) < 0 \\ n_v, & \text{if } g_j(\mathbf{x}) \geq 0 \text{ and } n_v > 0 \\ f(\mathbf{x}) & \text{otherwise} \end{cases}$$

where n_v is the number of violated constraints. Each sub- population will try to reduce the violation measure of the corresponding constraint as the first objective, and then the total number of constraint violations as the second objective. If the solution $(\mathbf{x})$ is feasible, then it will be evaluated with the objective function. The implementation of this method has to determine the size of each sub-population, which is itself an open issue.

Method of Coello and Mezura

Unlike the previous method, the second multi-objective method proposed by Coello *et al.* to handle constraints is based on the Pareto front [COE 02c]. However, the selection process is a hybrid mechanism using both Pareto-based selection and probabilistic selection. The method implements a version of the Niched-Pareto Genetic Algorithm (NPGA) but without niching. To maintain diversity in the population, only a ratio, S_r, of individuals are selected through dominance-based tournament selection (where S_r is a user-defined parameter). The remaining individuals are selected with a probabilistic approach.

Coello and Mezura introduced four comparison rules for the dominance-based tournament selection:

1) Any feasible individual is better than any unfeasible individual;

2) Among two feasible solutions, the one with the best fitness is preferred;

3) When both are unfeasible, the non-dominated solution in selected;

4) When both are unfeasible and non-dominated or dominated, the individual with the lower violation measure is preferred.

The proposed approach performed well in several test problems [COE 02c]. However, as is the case for several methods in the same category, with the proposed scheme, it is hard to maintain the population diversity, even with the probabilistic selection.

Method of Aguirre et al.*: IS-PAES*

As the previous method of Coello and Mezura, Aguirre *et al.* proposed a multi-objective method to handle constraints based on the Pareto-dominance selection [AGU 04]. The method is an extension of the Pareto Archived Evolutionary Strategy (PAES) [KNO 99]. It adds to the original scheme two crucial features:

– shrinking the objective space which reduces the size of the search space continuously by exploring the information of individuals surrounding the feasible region. Thus, the size of the search space will be very small and the solutions obtained will be competitive in the end;

– inverted "ownership" that handles the population as part of a grid location relationship by building a sorted list of the most densely populated areas of the grid.

The multi-objective concept is used in this case as a secondary criterion (Pareto dominance is used only to decide whether or not a new solution is inserted in the external memory).

Method of Cai and Wang

Cai and Wang proposed to handle constraints in the CEO with a multi-objective technique [CAI 06] extended with an Infeasible Solution Archiving and Replacement mechanism (ISARM). This mechanism stores in an archive $\mathbf{A}$ the best unfeasible individuals in the offspring population (having the lowest violation measures) where all the offspring population is unfeasible. The pseudo-code of this procedure is summarized by Algorithm 4.

```
$\mathbf{A}$: archive
$\mathbf{P}$: population
$g$: current generation
$m, n$: user defined parameters
$\hat{x}$: best unfeasible individual

if fitness values of feasible solutions are very close then
    if there is no feasible individual in the current offspring population
    then
        $\mathbf{A} \leftarrow \mathbf{A} \cup \hat{x}$
    end
    if $\mathrm{mod}(g, m) = 0$ then
        randomly choose at most $n$ individuals from the archive $A$ and
        replace $n$ individuals randomly selected from the population $\mathbf{P}$
        $\mathbf{A} \leftarrow \emptyset$
    end
end
return $\mathbf{A}, \mathbf{P}$
```

Algorithm 4: procedure ISARM($\mathbf{A}, \mathbf{P}, g, m, n$)

This mechanism might be useful for constrained problems with very small feasible region; however, it does not really help the search process when the algorithm is able to find easily feasible solutions.

3.6. Parallel population approaches

The idea of using a parallel population mechanism to handle CEO is the most recent approach. It aims to evolve several populations according to an ensemble of constraint-handling techniques. Parallel populations approaches can implement either Fine-Grained parallelism, where sub-populations communicate frequently along evolution, or Coarse-Grained parallelism, where communication between sub-populations is limited. Some recent methods to handle constraints implementing Fine-Grained parallelism were proposed during the last decade. They are summarized in the following paragraph.

Fine-Grained parallelism

An early method that has been proposed in this category is the ECHT (Ensemble of Constraint-Handling Techniques) [MAL 10]. Motivated by the no free lunch theorem [WOL 97], the authors assumed that it is impossible for a single constraint-handling technique to outperform all other techniques on every problem and that the efficiency of these techniques depends on several factors (ratio between feasible search space and search space, multimodality of the problem, stages of the search process, etc.). They then proposed to combine four constraint-handling techniques: feasibility rules, stochastic ranking, a self-adaptive penalty function and the ε-constrained method. Each of these methods has its own population. Sub-populations share all of their offspring, and all offspring are evaluated with all four different constraint-handling methods, i.e. an individual disregarded by its sub-population may survive in another population. The experiments have shown that ECHT performs better than any of the constraint-handling techniques that it contains.

Elsayed *et al.* [ELS 14] proposed to use with a Differential Evolution-based algorithm and in a parallel way, two constraint-handling techniques (feasibility rules and the ε-constrained method), four DE-mutations and two DE-recombinations. Initially, each individual is assigned a random combination of operators, and the improvement made in the population by each combination is recorded. Afterward, the best combination is assigned to more and more individuals, while each of the other individuals are assigned a random combination. The approach was tested on a set of 18 test problems with 10 and 30 dimensions, showing a very competitive performance.

3.7. Hybrid methods

The methods in this category handle the constraints by using various approaches, while the objective function is optimized with an evolutionary algorithm. There are two ways to accomplish this separation. The first one is to handle the constraints with a deterministic procedure for numerical optimization combined with the evolutionary algorithm. In the second approach, the evolutionary algorithm is coupled with another heuristic to handle constraints.

In the first approach, we can cite as an example the method of Myung [MYU 98], which extends the evolutionary algorithm using the Lagrange multipliers. As an example of the second approach, we consider the method of Leguizamon and Coello [LEG 07] that uses Ant Colonies to explore the boundaries of the feasible domain. Several currently used methods for the same approach have been published in the last decade. A more detailed description can be found in [MEZ 09]. Note also that many authors proposed hybrid techniques for CEO, but the combined technique is often used to maximize the population diversity or to enhance the convergence speed like for DEPSO and HPSO presented in section 3.3.2. For these two methods, the constraints are handled with the feasibility rules. Other hybrid methods use a penalty function or a multi-objective approach to handle constraints.

3.8. Conclusion

This chapter has presented a set of approaches to handle the constraints of an optimization problem which can be solved using evolutionary algorithms. The choice of an appropriate technique depends on several criteria mainly related to the nature of the problem. Several questions need to be asked in this context:

– is the objective function defined in the unfeasible domain? If not, several techniques cannot be applied, such as the penalization methods;

– are there some active constraints at the optimum? If there is no active constraints at the optimum, all the methods based on the search on the feasible region boundaries are irrelevant;

– what is the nature of the constraints? For example, if at least one of the constraints is a nonlinear inequality, the methods which handle only linear constraints are excluded, such as the GENOCOP system;

– is the ratio $|\mathcal{F}|/|\mathcal{S}|$ of the feasible space to the search space measures too small? If the problem has equality constraints or if this ratio is too small, it is preferable to avoid methods that have shown weak performance in this case, such as some approaches based on the superiority of the feasible solutions.

Some other criteria should also be considered for choosing a method, such as its ability to solve benchmark problems. Indeed, the performance of some approaches has been reported in several comparative studies. They may provide guidance on the choice of a good technique. However, the effectiveness of a method is often dominated by two other decision criteria that are the complexity and difficulty of implementation. For example, stochastic ranking (section 3.2.5), feasibility rules (section 3.3.2) and, more generally, the penalty methods (section 3.2) are the most popular constraint-handling approaches currently used, thanks to their simplicity [MEZ 11].

In conclusion, there is no general approach for handling the constraints with evolutionary algorithms able to deal with any type of problem. This subject continues to be the focus of numerous research projects in the field.

4

Combinatorial Optimization

4.1. Introduction

Combinatorial optimization can be defined as "*the mathematical study of finding an optimal arrangement, grouping, ordering or selection of discrete objects usually finite in numbers*" [OSM 12].

Many real-life problems can be formulated as combinatorial optimization problems in different areas, such as in production (e.g. production speed up, cost reduction, efficient utilization of resources), transportation (e.g. fuel saving, reduction of delivery time), employee scheduling (e.g. reduction of human resources), hardware design (e.g. acceleration of computations) and communication network design (e.g. end-to-end delay reduction).

Formally, a combinatorial optimization problem is a triple $(\Omega, \mathcal{F}, f)$, where:

– Ω is a finite search space;

– $\mathcal{F} \subseteq \Omega$ is the feasible domain;

– f is the objective function defined on $\mathcal{F}$.

A combinatorial optimization problem can be either *easy* or *hard*. A problem is said to be *easy* if there is an efficient algorithm to solve for optimality all of its instances in polynomial time. Otherwise, a problem is said to be *hard*. The huge computational costs of solving hard problems have led researchers to develop heuristic algorithms, such as Genetic and

Evolutionary Algorithms (GEAs), to obtain near-optimal solutions much faster than optimal solutions with an exact method.

There are numerous examples for the successful application of GEAs to combinatorial optimization problems. Many published books, such as those written by Davis [DAV 91] and Chambers [CHA 98], display the range of problems to which GEAs have been applied. This chapter presents the basic concepts of the application of GEA to combinatorial problems.

Using GEAs for solving combinatorial problems needs at first to determine how to encode solutions and how to evolve solutions. In the 1980s, some researchers aimed at tackling the Traveling Salesman Problem (TSP) by using GEAs based on the binary alphabet. However, this representation is useful when the solutions can be naturally encoded with a binary string such as for the knapsack problem; however, it has serious difficulties when handling heavily constrained problems. In 1985, Goldberg and Lingle [GOL 85] proposed a different representation: the *Order-Based Representation* (OBR). It is based on the relative order of the symbols in the genotype. OBR can be used to solve a great variety of combinatorial problems (scheduling problems, graph coloring, ordering genes, etc.). That is why, later, several works in the domain proposed solutions to develop and improve operators for OBR, using the Traveling Salesman Problem as a reference application. Other representations were also proposed for more complex or specific problems.

4.1.1. *Solution encoding*

There exist many ways to encode solutions for combinatorial optimization problems. The encoding technique depends on the problem and essentially on the information that should be exchanged between individuals. So, the first step to design the solution encoding is to define the kind of information that has to be exchanged. This is considered in a higher level. The second step, in a lower level, is the way to perform this exchange. Then, an encoding design is defined. It is important to define, at the same time, the exchange methods implemented by the crossover operators.

Depending on the encoding method and the variation operators, the algorithm might generate unfeasible offspring that cannot be considered as solutions of the problem. Three strategies are possible to deal with unfeasible solutions: repairing, penalizing and killing. Repairing consists of transforming solutions so as to present a valid solution of the problem. Killing consists simply of rejecting unfeasible solutions from the offspring population and replace them by new individuals. Penalization consists of adding a penalty to the fitness value of the unfeasible solutions. Note that often, EAs for combinatorial optimization use convenient encoding and variation operators that avoid the creation of unfeasible solutions.

Solutions of many combinatorial optimization problems can be represented using one of the following encoding techniques:

– Binary encoding: a solution is encoded with a binary string using a binary alphabet $\{0, 1\}$;

– Order-based encoding: In this scheme, a solution is a sequence of n ordered objects. A genotype might be a string of numbers that represent a position (number) in the sequence. Given n unique objects, $n!$ permutations of the objects exist;

– Other encoding schemes: many other representations were proposed depending on the problem considered. In this case, since the binary operators and the Order-based operators yet introduced in the literature are not be used, it is important to specify the pair (encoding, operator definitions).

Many combinatorial problems can be handled using binary encoding (e.g. knapsack problem, SAT problems, Set-Covering problems) or Order-based encoding (e.g. Traveling Salesman Problem, Scheduling problems, Vehicle routing problem). To evolve genotypes based on binary encoding, we can use the binary operators presented in Chapter 1 of this book. However, they can produce invalid solutions. In this case, repairing methods are needed. Otherwise, special operators have to be defined to evolve order-based chromosomes. Many variation operators have been proposed since the 1980s. This chapter presents commonly used crossover and mutation operators for the TSP problem as an example of application.

4.1.2. *The knapsack problem (KP)*

The knapsack problem or rucksack problem is stated as follows: given a set of items, each with a weight and a value (benefit), determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

Formulation

The most common problem is the 0–1 knapsack problem, which restricts the number (x_i) of copies of each kind of item to zero or one. Given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity W, the 0-1 KP may be formulated as follows:

$$\text{Maximize } f(x) = \sum_{i=1}^{n} v_i x_i$$

subject to $\sum_{i=1}^{n} w_i x_i \leq W$ and $x_i \in \{0,1\}$

Here, x_i is a binary variable equal to 1 if the item i should be included in the knapsack and 0 otherwise. Informally, the problem is to maximize the sum of the values of the items in the knapsack so that the sum of the weights is less than or equal to knapsack's capacity W.

Feasibility: A feasible solution for the 0-1 KP is a vector $X = (x_1, x_2, \ldots, x_n), x_i \in \{0,1\}$ such that $\sum_{i=1}^{n} w_i x_i \leq W$. Because of this constraint, the 0-1 knapsack problem is NP-hard.

Example

Let us consider a knapsack having as maximum capacity 28 cubic inches and a set of items with different sizes and different benefits. There are six potential items (labeled "A","B","C","D","E","F"), whose volumes and benefits are as follows:

Item	A	B	C	D	E	F
Benefit	4	3	7	8	5	6
Volume	7	8	6	5	4	9

We seek to find the optimal solution $X = (x_1, x_2, x_3, x_4, x_5, x_6)$ that maximizes the total benefit computed by:

$$f(X) = 4x_1 + 3x_2 + 7x_3 + 8x_4 + 5x_x + 6x_6$$

and satisfies the constraint: $7x_1 + 8x_2 + 6x_3 + 5x_4 + 4x_x + 9x_6 \leq 28$, where $x_i \in \{0, 1\}$ for $i = 1, 2, 3, 4, 5, 6$. For this problem, there are 2^6 possible solutions.

4.1.3. *The Traveling Salesman Problem (TSP)*

The traveling salesman problem is stated as follows: given a number (n) of cities with associated city-to-city distances, what is a shortest round trip tour that visits each city exactly once and returns to the start city? The task is to arrange a tour of n cities such that each city is visited only once and the length of the tour is minimized. The number of solutions in the search space grows as $(n-1)!/2$, where n is the number of cities (e.g. for seven cities, there are 360 possible solutions in the search space, and for 100 cities, there are $4.67 \cdot 10^{155}$ possible solutions). The optimization of the traveling salesman problem continues to receive attention for three reasons:

1) it is an NP-hard problem although it is easily expressed;

2) it serves as the simplest case of many real-world problems;

3) it has become a *benchmark* problem for comparing methods.

Formulation

Let $G = (V, A)$ be a graph, where V is a set of n vertices (cities). A is a set of arcs or edges, and let $C : (C_{ij})$ be a distance (or cost) matrix associated with A, which gives all distances $d(v_i, v_j)$ with $i, j = 1, \cdots, n$, between each pair of cities. This matrix is *symmetric* if the distance between any pair of cities is unchanged in either direction. In this case, G is a non-oriented graph. If the matrix is asymmetric, $d(v_i, v_j) \neq d(v_j, v_i)$ and G is an oriented graph.

The TSP consists of determining a minimum distance circuit passing through each vertex once and only once. In other words, given a sequence $(v_1, v_2, \cdots v_n)$ of the cities and intercity distances, $C_{ij} = d(v_i, v_j)$, the goal is to find a permutation Π of the cities that minimizes the sum of distances:

$$\text{minimize } f(\Pi) = \left[\sum_{i=1}^{n-1} d(\Pi(i), \Pi(i+1))\right] + d(\Pi(n), \Pi(1))$$

where $(\Pi(i), \Pi(i+1))$ are two adjacent cities in the path Π and $d(\Pi(n), \Pi(1))$ is the distance between the final city and the start city, which must be added to the total distance since the circuit of the TSP is a round trip.

Example

For example, let us consider a set of 10 cities $V = \{v_1, ..., v_{10}\}$ numbered from 0 to 9. A path is defined by a sequence of integer numbers associated with the cities. For the Order-Based Representation, a genotype is simply this sequence. For example, the path

$$7 \to 1 \to 0 \to 6 \to 2 \to 5 \to 8 \to 9 \to 3 \to 4 \to 7$$

is represented by the string (7 1 0 6 2 5 8 9 3 4) (Figure 4.1).

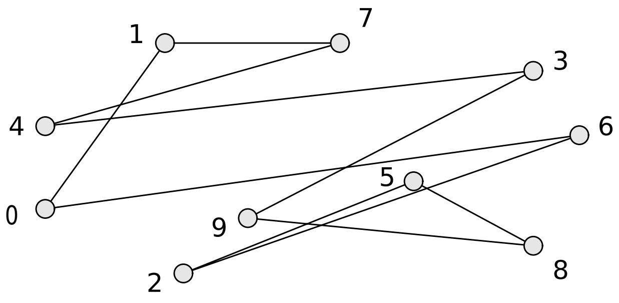


Figure 4.1. *Example of a feasible tour for a TSP with 10 cities.*

4.2. The binary representation and variation operators

Several combinatorial problems can be naturally encoded with the binary representation. In this case, there is no distinction between genotype and phenotype. All variation operators defined for the binary representation might be used to solve such problems. For example, the three different forms of crossover, one-point crossover, N-point crossover and uniform crossover, are suitable for binary encoding (see Chapter 1, section 1.5.1). In the same way, the mutation operators generally used for the binary encoding, such as the bit flip mutation, might also be applied in solving combinatorial optimization problems with GEAs. Only a few additional operators were proposed to improve the global performance of GEAs. Below we present an example.

Beasley and Chu [BEA 96] proposed a specific crossover operator called the *fusion-operator* for the Set-Covering problem. It takes into account both the structure and the relative fitness of the parents for a minimization. From two given parents, p_1 and p_2, it produces a unique child according to the steps described in Algorithm 1. The *fusion-operator* assumes that the inheritance of symbols responsible for a good fitness value of a parent are more likely to contribute to a good fitness for the child. Beasley and Chu also proposed a *heuristic feasibility operator* [BEA 96], but it is specific to the Set-Covering problem. It could be considered as a local search procedure to improve solutions in the sense of reducing the over-covered constraints. This heuristic is suitable for small size problems.

```
$f(p_1)$, $f(p_2)$: fitnesses of the parents $p_1$ and $p_2$ respectively
$C$: resulting child
for all $i \in \{1, ..., n\}$ do
    if $p_1[i] = p_2[i]$ then
        $C[i] \leftarrow p_1[i] \leftarrow p_2[i]$
    end
    else
        $p \leftarrow \frac{f(p_2)}{f(p_1)+f(p_2)}$
        if $p > \mathcal{U}[0,1]$    // $\mathcal{U}[0,1]$: uniform random draw in [0, 1]
        then
            $C[i] \leftarrow p_1[i]$
        end
        else
            $C[i] \leftarrow p_2[i]$
        end
    end
end
return $C$
```

Algorithm 1: Function Fusion_Operator(p_1, p_2)

It is easy to implement a GEA program to solve binary encoded combinatorial optimization problems. However, a standard crossover operator may not produce feasible offspring from two feasible parents. To handle constraints, specific crossover operators may be designed, unless penalization or multi-objective techniques are implemented.

4.2.1. *Binary representation for the 0/1-KP*

The binary representation is well suited for the 0/1-KP problem. Indeed, a solution is simply encoded with a binary genotype of size n, where n is the number of items in the problem. For the example presented in section 4.1.2, a genotype can be represented by a string of 6 bits. Thus, the genotype (1 0 0 1 0 0) denotes that only items A and D are added to the knapsack, and it is a feasible solution since $7 + 5 = 12 \leq 28$.

To handle constraints in the 0/1-KP problem, the most common practice is to use a penalization technique that adds a penalty to the fitness of the unfeasible solutions (see Chapter 3) [KHU 94]. Another approach consists of transforming the problem into a multi-objective problem, where minimizing the sum of weights becomes the second objective with the main objective of maximizing the profit:

$$\text{Maximize } f(x) = \sum_{i=1}^{n} v_i x_i \text{ and Minimize } \sum_{i=1}^{n} w_i x_i$$

Note that the multi-objective variant of the problem can be harder than the single-objective case because the number of points in the Pareto set can be exponentially large [KUM 06].

4.2.2. *Binary representation for the TSP*

With a binary representation of the n-cities TSP, each city is encoded as a string of $\log_2 n$ bits, an individual is then a string of $n[\log_2 n]$ bits. For example, in the 6-cities TSP, the cities can be represented by 3-bit strings. Then, the tour $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 1$ is represented by an 18-bit string: $(000\ 001\ 010\ 011\ 100\ 101)$. Note that there exist 3-bit strings which do not correspond to any city: the strings are (110) and (111).

When applying the classical binary variation operators, offspring might not represent a legal tour. A *repair algorithm* is then needed to change offspring into legal solutions [LAR 99]. Because of these difficulties, the binary representation is not considered to be appropriate for the TSP, as commented by Whitley *et al.* [WHI 89]:

Unfortunately, there is no practical way to encode a TSP as a binary string that does not have ordering dependencies or to which operators *can be applied in a meaningful fashion. Simply crossing strings of cities produces duplicates and omissions. Thus, to solve this problem, some variation on standard genetic crossover must be used. The ideal recombination operator should recombine critical information from the parent structures in a non-destructive and meaningful manner*

4.3. Order-based Representation and variation operators

The Order-based Representation is the most used encoding design in evolutionary combinatorial optimization to solve permutation problems, such as the Traveling Salesman Problems, the Vehicle Routing Problems and graph coloring.

If mutation and crossover operators, defined for binary or integer representations such as uniform crossovers or point mutations, are used to evolve order-based genotypes, the resulting offspring represent invalid solutions in the search space. For example, the uniqueness constraint of a city in a feasible tour for the TSP can be violated. For this reason, many generic operators for evolving permutations were proposed. We selected the most used ones in the literature [LAR 99].

Other alternatives use specific operators dedicated to given combinatorial problems. Some examples are given in the section 4.3.3.

4.3.1. *Crossover operators*

Many crossover variants have been proposed in the literature for the Order-based Representation, such as:

– the uniform order-based crossover [SYS 89];

– the order crossover [DAV 85, OLI 87, DEE 11];

– the partially mapped crossover [GOL 85];

– the cycle crossover [DAV 85, OLI 87];

– the voting recombination crossover [MÜH 89];

– the heuristic crossover for adjacency representation [GRE 85];

– the maximal preservative crossover [MÜH 88];

– the alternating edge crossover [GRE 85];

– the genetic edge recombination crossover [WHI 89];

– the inver-over crossover [TAO 98].

Some crossover variants only need the position of elements in the parent genotypes to produce an offspring. In addition, others use the adjacency information between different elements to decide which sequence to exchange. Below, we present in detail each operator in this list.

4.3.1.1. *Uniform Order-based Crossover (POS)*

To cross two parents, $P1$ and $P2$, the uniform order-based (or position based) crossover [SYS 89] generates a random binary template. The first offspring takes the genes from $P1$ having a position corresponding to the "ones" in the template. The other genes of $P1$ are then sorted in the same order as they appear in $P2$, and they are used to fill the gaps in *offspring1*. The second offspring is created using the same process by inverting the roles of parents $P1$ and $P2$.

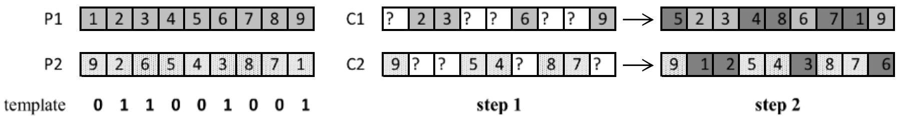


Figure 4.2. *Example of Uniform Order-based Crossover*

4.3.1.2. *Order-based Crossover (OX)*

The order-based crossover is a variation of the Uniform order-based crossover introduced for the purpose of preserving the relative order of symbols in the sequences to be combined. It was proposed by Davis [DAV 85] mainly for scheduling problems. Instead of using a binary template, the OX operator generates two cut points for both parents. The symbols between the two cut points in $P1$ and $P2$ are then copied to the children $C1$ and $C2$, respectively. The sequence of symbols used to fill $C1$ is obtained from $P2$ starting from the second cut point, omitting those which already exist in $C1$. Reaching the end of $P2$, the copy continues from its first symbol. The same operation is done for the second offspring by inverting the role of parents. Figure 4.3 presents an example of OX crossover. To create the first offspring $C1$, the elements 3, 4, 5 and 6 are inherited from $P1$ in the order and position in which they occur in $P1$. Then, starting from the second cut point, the child inherits from the second parent $P2$. The sequence of symbols in $P2$ (from the second cut point) is (8 7 4 9 2 6 5 1 3). After removing the already existing elements in $C1$, the sequence becomes (8 7 9 2 1). This sequence is placed in $C1$ starting from the second cut point. The same operation is done for the second child by inverting strings.

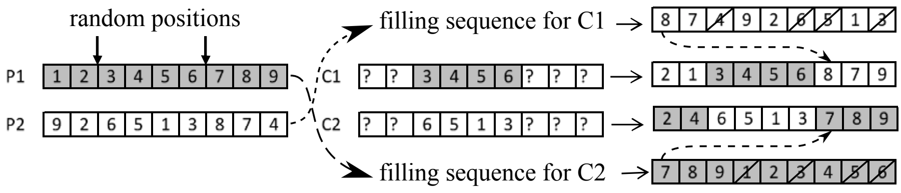


Figure 4.3. *Example of Order-based Crossover*

Davis proposed a second variant of OX, denoted by OX_2. It differs from the original OX in the repairing method. The first step is the same where the elements between two cut points are copied to offspring. The remaining elements are copied from the beginning of the second parent with respect to their relative order.

Considering the same example given in Figure 4.3, let us define the first offspring from the second step. The sequence of symbols in the second parent (from the beginning of the string) is (9 2 6 5 1 3 8 7 4). After removing the already existing symbols, the sequence becomes (9 2 1 8 7). This sequence is placed in the first offspring from left to right according to the original order. The same process is done for the second child by inverting the role of parents. The produced offspring are then:

$C1$ (9 2 | 3 4 5 6 | 1 8 7)

$C2$ (2 4 | 6 5 1 3 | 7 8 9)

Deep and Mebrahtu [DEE 11] proposed three other variations of OX. In the first variant, OX_3, the cut points in parent 1 and parent 2 are at different positions, but the size of the sub-string between the cut points is the same for both parents. The symbols are copied to the offspring respecting the same rules defined for the original OX. Then, if we consider the precedent example with a first cut point in position 2 for parent 1 and position 4 for parent 2, and a sub-string of size 4, then the generated offspring after the first step are:

$P1$: (1 2 | 3 4 5 6 | 7 8 9), $C1$: (? ? | 3 4 5 6 | ? ? ?)

$P2$: (9 2 6 5 | 1 3 8 7 | 4), $C2$: (? ? ? ? | 1 3 8 7 | ?)

Starting from position 9 after the second cut point in $P2$, the filling sequence for $C1$ is (9 2 1 8 7). Similarly, starting from the position 7 after the second cut point in $P1$, the filling sequence for $C2$ is (9 2 4 5 6). These sequences are copied in the corresponding offspring usually starting from the second cut point. The final offspring are then:

$C1$: (8 7 | 3 4 5 6 | 9 2 1)

$C2$: (9 2 4 5 | 1 3 8 7 | 6)

This variant can be also applied with a different sub-string size (OX_4). Deep and Mebrahtu [DEE 11] also proposed a fifth variant that applies the same steps as OX but uses two pairs of cut points (OX_5). For example, let us consider the previous example with four cut points as follows:

$P1$ (1 | 2 3 | 4 5 6 | 7 8 | 9)

$P2$ (9 | 2 6 | 5 1 3 | 8 7 | 4)

Sub-strings between the two pairs of cut points are copied in the offspring:

$C1$ (? | 2 3 | ? ? ? | 7 8 | ?)

$C2$ (? | 2 6 | ? ? ? | 8 7 | ?)

Then, each offspring is filled using a sequence of symbols created from the alternate parent starting from the last cut point using the same rules as for OX_1. The final offspring are then:

$C1$ (9 | 2 3 | 6 5 1 | 7 8 | 4)

$C2$ (1 | 2 6 | 3 4 5 | 8 7 | 9)

The authors test the five implementations of OX on six TSP benchmarks, with the number of cities varying from 51 to 195. They demonstrated that OX_4 and OX_5 perform better for the number of cities greater than 100.

4.3.1.3. *Partially Matched (or Mapped) Crossover (PMX)*

PMX was proposed by Goldberg and Lingle [GOL 85] for the Order-based representation to solve the TSP problem. The PMX crosses two parents and ensures the uniqueness of symbols in the offspring. To accomplish these tasks, PMX operates in two main steps: *swapping* and *mapping*, as described in the following:

1) *Swapping*: This first step is quite similar to the OX. Two crossover points are selected randomly. The components between these two points define the *matching* section. The corresponding crossover points are then reproduced on the second parent. Next, sub-strings between the two crossover points are exchanged to create two incomplete offspring (Figure 4.4, step 1).

2) *Mapping*: In the second step, each offspring is completed using the remaining elements in the corresponding parent (the i^{th} offspring is filled with the remaining elements of the parent i). If the element is already present, then a correction is performed according to the mapping defined above.

For instance, let us consider the parents $P1$ and $P2$ and the cut points given in Figure 4.4. Here, * denotes a position that is not filled yet.

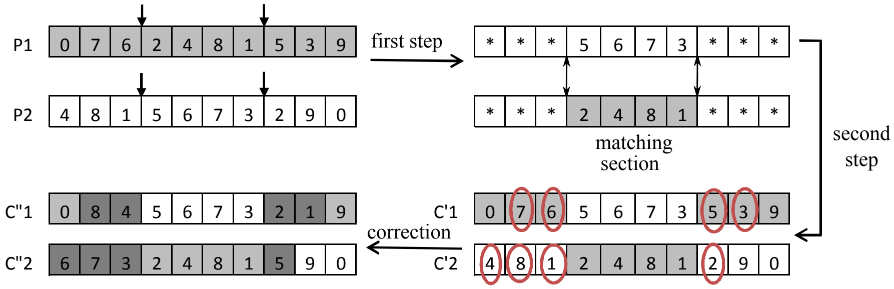


Figure 4.4. *Example of PMX*

The first step based on the *matching* section, starts by defining the following mapping:

$$5 \leftrightarrow 2; 6 \leftrightarrow 4; 7 \leftrightarrow 8; 3 \leftrightarrow 1$$

In the second step, the vacant parts of $C1$ (respectively $C2$) are filled with the remaining elements from $P1$ (respectively $P2$) that happen to be in the same position. We obtain $C'1$ and $C'2$. However, four elements in each offspring are duplicated: 7,6,5,3 in $C'1$ and 4,8,1,2 in $C'2$. These elements are then replaced according to the mapping previously defined in the first step. This produces offspring C''_1 and C''_2 that have no duplicated element.

The PMX crossover is very popular and has been applied extensively to the TSP and similar problems.

4.3.1.4. *Maximal Preservative Crossover (MPX)*

The *maximal preservative* operator was introduced by Muhlenbein *et al.* in 1988 [MÜH 88]. It works in a similar way to the PMX operator; however, it uses some additional restrictions for the sub-string size. Indeed, to assure that there is enough information exchange between the parents without losing too much information, the size of the sub-string selected from the first parent must be:

– greater than or equal to 10 (except for very small problem instances);

– smaller than or equal to the problem size divided by 2.

The first section of the offspring is the sub-string selected from first parent. Next, all the copied symbols are removed from the second parent and the remaining section of the offspring is filled with symbols in the same order as they appear in the second parent. If we consider the same initial genotypes as in the previous example, the sub-string size in the first parent must be less than or equal to 5. Figure 4.5 shows the produced offspring with a sub-string of four symbols starting from position 4.

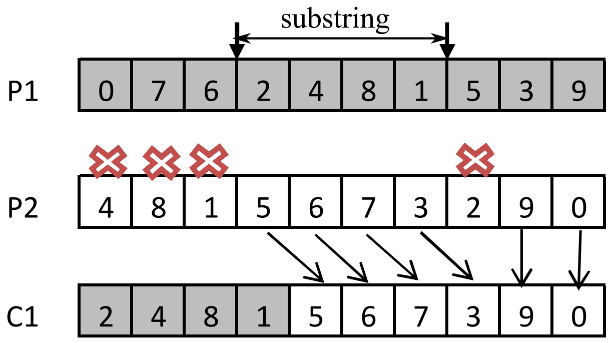


Figure 4.5. *Example of MPX*

4.3.1.5. *Cycle Crossover (CX)*

The first implementation of cycle crossover was proposed by Oliver *et al.* in 1987 [OLI 87]. It works like the uniform crossover with the difference that the symbols from the parents are not selected randomly. The position of an element to be copied from the first parent to the child is defined by the second parent. For instance, let two parents $P1$ and $P2$ be the same as in the previous example (Figure 4.6). We start by copying the first element from $P1$ to the child c1 (i.e. 0). Next, we select the symbol from $P2$ having the same position as 0 in $P1$ (that is 4). We copy this symbol into C1 but at the position where it appears in $P1$ (i.e. position 5). Next, we select the symbol from $P2$ having the same position as the last copied symbol from $P1$ (i.e. 6) and it is copied in c1 at the position where it can be found in $P1$ (position 3). We proceed in the same way until an unfeasible element is selected (already present in the child). At this step, the cycle is finished. Then, we complete all remaining position in the child with the symbols from $P2$ at the same positions. Note that we can create another child C2 by reversing the roles of the parents.

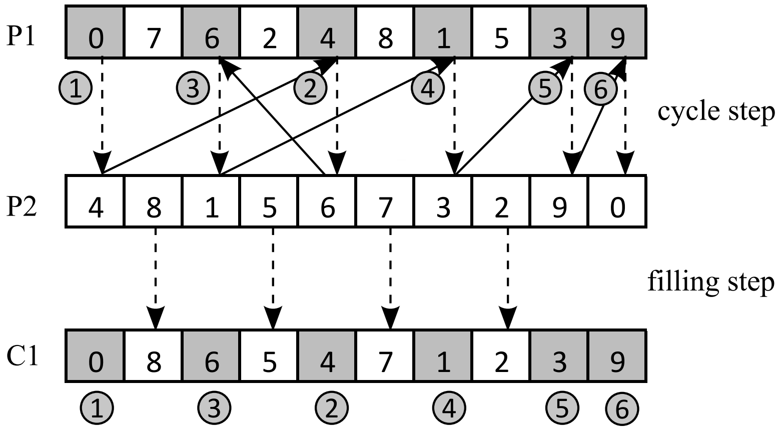


Figure 4.6. *Example of Cycle Crossover*

4.3.1.6. *Voting Crossover (VR)*

The Voting Crossover was introduced in 1989 by Muhlenbein [MÜH 89], who named it "p-sexual Voting Recombination" (VR), where p is a natural number greater than or equal to 2. VR is a multi-parent crossover. It needs two parameters: p is the number of parents to be selected for recombination and

$\alpha < p$ is a threshold value that determines the minimum number of parents that must have the same symbol in the same position to accept the vote. Let us take an example of five parents illustrated in Figure 4.7 and a threshold set to 3. To define a symbol to copy in the child at each position, a voting is proceeded. If the symbol appears three or more times in different parents and in the same position, then it is copied in the offspring usually in the same position. After the voting step, symbols 0, 6, 1 and 9 are copied in the offspring. The remaining positions are filled during the mutation step that performs a random selection from the not yet selected symbols.

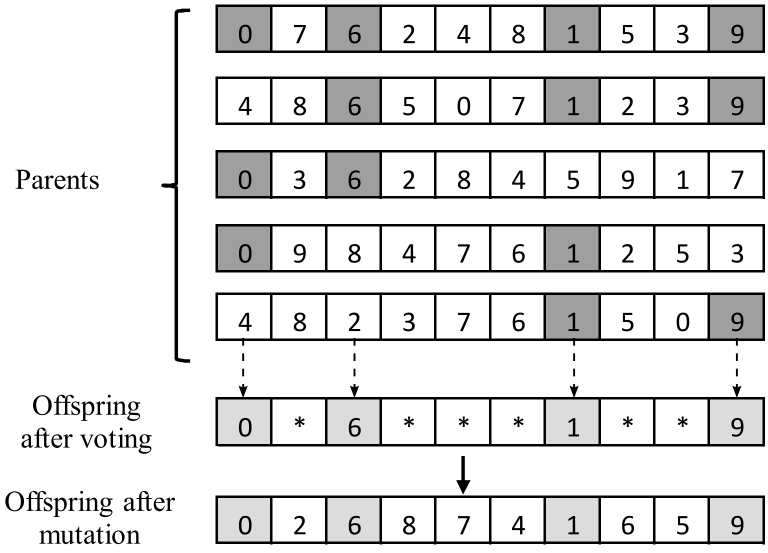


Figure 4.7. *Example of Voting Crossover*

4.3.1.7. *Alternating Edges Crossover (AEX)*

In 1985, Grefenstette *et al.* proposed the Alternating Edges Crossover (AEX) [GRE 85] that considers a genotype as a directed cycle of arcs, where an arc is defined by two successive symbols. To create a child from two parents, arcs are chosen in alternation from the two parents, with some additional random choices in case of unfeasibility.

Figure 4.8 illustrates the AEX behavior with two parents $P1$ and $P2$ similar to the two previous examples of VX and CX. AEX starts by choosing the arc

($0 \rightarrow 7$) from the first parent $P1$. Next, the arc having as first symbol 7 in the parent $P2$ is selected (arc 2) and the second symbol (i.e. 3) is added to the child. AEX proceeds in the same way until the child is complete or an unfeasible arc is selected. For instance, the fourth arc selected in the example of Figure 4.8 is unfeasible (0 is already present in the child). Then, AEX selects randomly a symbol from no-visited symbols as a starting point for the following arc and continues to fill the child with the ordinary procedure. The last arc is also an unfeasible arc, however, since there is only one remaining symbol (4), it is used automatically to complete the offspring.

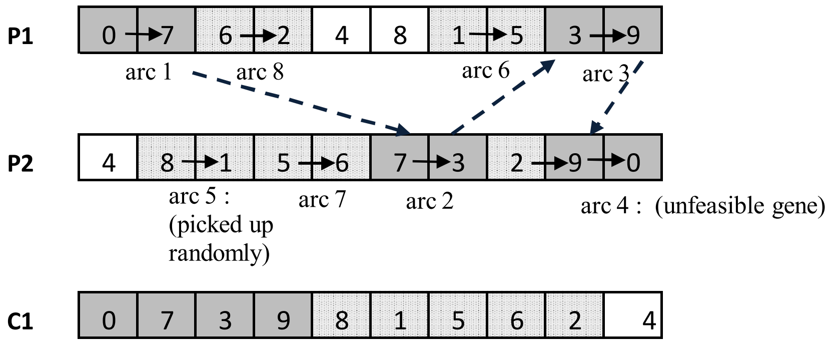


Figure 4.8. *Example of Alternating Edges Crossover. Arc_i is the selected arc for the offspring at the i^{th} step. The arc four leads to an unfeasible symbol (0 is already present in the offspring, then the starting symbol of the following arc (arc 5) is selected randomly from no-visited symbols*

4.3.1.8. *Heuristic Crossover (HGreX, HRndX, HpPoX)*

The *Heuristic crossover* (HGreX) was proposed by Grefenstette [GRE 87] for the TSP. As AEX, the heuristic crossover considers the genotype as a directed cycle. HGreX compares two arcs of the parents and selects the cheapest one to add to their child. In case of unfeasibility, some additional steps are done. Let us consider the same parents, $P1$ and $P2$, as in the AEX example. To apply the HGreX, we need also the costs of all the arcs in each parent. For this example, Figure 4.9 shows the lists of costs for $P1$ and $P2$.

HGreX starts by selecting a random position in $P1$, which is a vertex v of the cycle. Then, it considers the two arcs in $P1$ and $P2$ for which v is their origin vertex. For example, let 7 be the starting vertex for arcs $7 \rightarrow 6$ in $P1$ and $7 \rightarrow 3$ in $P2$. Since $C(7, 6) = 8$ and $C(7, 3) = 4$, arc $7 \rightarrow 3$ is the first arc in the child. The second pair of arcs to compare is $3 \rightarrow 2$ in $P2$ and $3 \rightarrow 9$ in $P1$ with costs 2 and 3, respectively. Then, the second arc selected is $3 \rightarrow 2$. By the same way, arcs $2 \rightarrow 4$, $4 \rightarrow 8$, $8 \rightarrow 1$ and $1 \rightarrow 5$ are added to the child. In step 7, when considering the arcs $5 \rightarrow 3$ in $P1$ and $5 \rightarrow 6$ in $P2$, the second one is picked up since the first arc is unfeasible: 3 is already added to the child. When both arcs are unfeasible, the operator randomly generates a prescribed number of feasible arcs starting from the same vertex and selects the cheapest one.

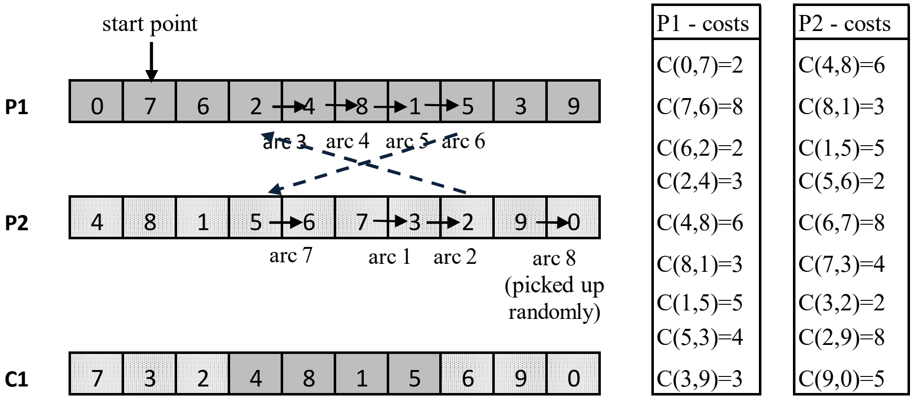


Figure 4.9. *Example of Heuristic Crossover: "arc i" is the selected arc for the offspring at the i^{th} step*

The operators HRndX and HProX are variants of HGreX [LAR 99]. They proceed on the same steps. However, when arcs of $P1$ and $P2$ are unfeasible from a given vertex, instead of choosing the feasible cheapest arc, HRndX makes random choices. Thereby, HRndX picks up any arc with equal probability, while HProX gives more chance to a cheaper arc.

4.3.1.9. *Genetic Edge Recombination (ERX)*

Genetic Edge Recombination was proposed by Whitley in 1989 [WHI 89] to solve the undirected TSP. ERX creates offspring by considering each solution in its argument as a collection of undirected edges and by trying to keep as many of the edges that occur with higher frequencies as possible. Its main objective is to give to the child a maximum of information about its parents, especially the state of their neighborhood.

Let us consider two parents: $P1 = (x_1, x_2, \cdots, x_n)$ and $P2 = (y_1, y_2, \cdots, y_n)$. For every element in $P1$ and $P2$, the operator first creates a set of neighbors in both $P1$ and $P2$. The collection of neighbor sets is called the edge-map. Of course, some of these neighbors can be identical. In this case, the cardinality of the set of neighbors of an element can be smaller than 4. For example, if $x_k = y_l$, then the set of neighbors of x_k and y_l will be $x_{k-1}, x_{k+1}, y_{l-1}, y_{l+1}$. Duplicate elements, if any, are removed. When the edge-map is built, ERX starts generating the offspring. It randomly selects a symbol x_c from the first parent that becomes the current symbol. Then, it adds x_c to the offspring and removes all its occurrences in the edge-map. To choose the next current symbol, it finds in the set corresponding to the x_c entry in the edge-map the element having the lowest set cardinality in the edge-map. This element is then added to the child, removed from the edge-map and becomes the new current element. The process continues until all entries are visited. If the neighbor set of x_c is empty and there is still some elements to consider, then another current symbol is chosen randomly from unvisited elements.

Let us explain ERX steps with an example. Considering the parents $P1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ and $P2 = (9\ 2\ 6\ 5\ 1\ 3\ 8\ 7\ 4)$, the corresponding edge-map will be as follows:

$$1 :\{5, 3, 2, 9\}4 : \{7, 9, 3, 5\}7 : \{8, 4, 6\}$$
$$2 :\{9, 6, 1, 3\}5 : \{6, 1, 4\}8 : \{3, 7, 9\}$$
$$3 :\{1, 8, 2, 4\}6 : \{2, 5, 7\}9 : \{4, 2, 8, 1\}$$

If 3 is chosen as the first element, then it is added to the offspring and removed from all the sets. The edge-map becomes:

$$1 :\{5, 2, 9\}4 : \{7, 9, 5\}7 : \{8, 4, 6\}$$
$$2 :\{9, 6, 1, \}5 : \{6, 1, 4\}8 : \{7, 9\}$$
$$3 :\{1, 8, 2, 4\}6 : \{2, 5, 7\}9 : \{4, 2, 8, 1\}$$

For the next position, we must consider 1, 8, 2 and 4, the neighbors of 3. Among them, 8 has the smallest number of neighbors, and hence it is the next element in the solution. The update of the edge-map and the choice of the next element are performed as for the previous symbol. This process is repeated until the offspring is complete. The ERX procedure is summarized in the following algorithm.

```
Build the edge-map
Choose randomly a starting element $x_c$ in $P1(x_1, \cdots, x_n)$
*stop* ← **false**
**repeat**
    Remove $x_c$ from the edge-map
    **if** *(cardinality of neighbor set of $x_c$) = 0* **then**
        **if** *there is unvisited elements* **then**
            choose $x_c$ randomly from unvisited elements
        **end**
        **else**
            *stop* ← **true**
        **end**
    **end**
    **else**
        choose $y_c$ from the neighbor set of $x_c$, where $y_c$ has the lowest
        neighbor set cardinality in the edge-map
    **end**
    $x_c \leftarrow y_c$
**until** *stop*
```

Algorithm 2: Genetic Edge Recombination procedure

According to Grefenstette *et al.* [GRE 85] and Larranaga *et al.* [LAR 99], the ERX crossover operator is the favored operator to solve the TSP with Evolutionary Algorithms.

4.3.1.10. *Inver-Over Operator (IOX)*

Tao and Michalewicz proposed the Inver-Over Operator [TAO 98] to evolutionary algorithms extended with local search for solving combinatorial problems. This operator has two adaptive components:

1) the number of inversions applied to each individual;

2) the parameters of sub-strings to invert can be randomly determined or can be determined by other individuals.

IOX starts by choosing a first parent $P1 = (x_0, x_1, \cdots, x_{n-1})$ and by selecting randomly the vertex x_c in $P1$. Next, for each inversion, it chooses another vertex $x_{c'}$ in $P1$. Then, it reverses the sub-string between positions c' and c in $P1$. c' is determined in two different ways:

– with probability p, which is a user-defined parameter, c' is chosen uniformly in $[0, n-1]$ such that $c' \neq c$;

– or else (with probability $1-p$), IOX chooses randomly a second parent $P2 = (y_0, y_1, \cdots, y_{n-1})$ from the population. Then, it determines the vertex $x_{c'}$ in $P1$ as the next vertex of x_c in $P2$.

Inversions are performed until $c' = (c+1) \bmod n$ or $c' = (c-1) \bmod n$. Algorithm 3 describes this procedure formally.

```
choose randomly a first element $x_c$ in $P1 = (x_0, \cdots, x_{n-1})$
repeat
    if $p > \mathcal{U}[0,1]$    // $\mathcal{U}[0,1]$:  uniform random draw in [0, 1]
    then
        choose uniformly $x_{c'} \in P1$, $c' \neq c$
    end
    else
        choose $P2 = (y_0, \cdots, y_{n-1})$ randomly in the population
        choose $c'$ such that $x_{c'} = y_{(d+1) \bmod n}$ where $d$ is such that
        $y_d = x_c$
                                                   // figure 4.10
    end
    reverse the substring between indexes $(c+1) \bmod n$ and $c'$ in $P1$
until $(c+1) \bmod n = c'$ or $(c-1) \bmod n = c'$
```

Algorithm 3: Inver-Over Operator procedure

4.3.2. *Mutation operators*

The mutation operator is often given little importance in research papers; however, it contributes to maintaining population diversity, to avoid local optima and also to improve exploration of the search space. As for crossover operators, the majority of the mutation operators were introduced first for the TSP problem. The most well-known mutation operators are:

insertion mutation [FOG 88, MIC 92], simple inversion mutation [HOL 75], inversion mutation [FOG 88], exchange mutation [OLI 87], repeated exchange mutation [AMB 91, BEY 92], displacement mutation [MIC 92] and scramble mutation [SYS 91].

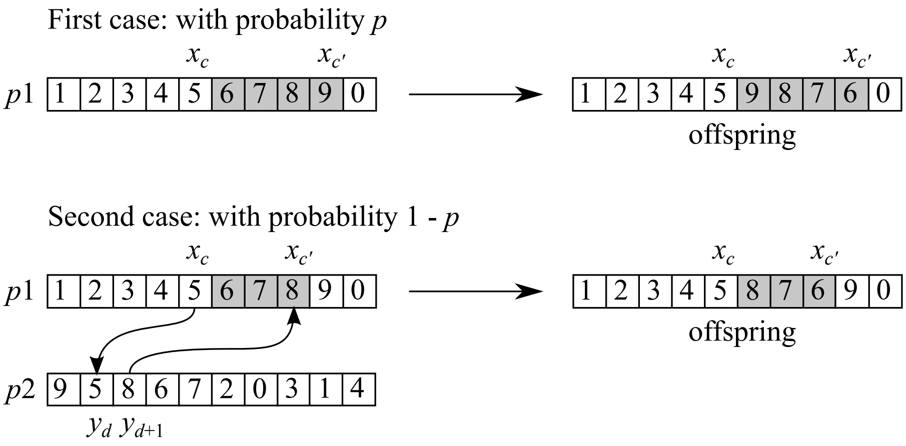


Figure 4.10. *One step of the Inver-Over crossover*

These operators are summarized in the following sections.

4.3.2.1. *Mutation by insertion (ISM)*

The insertion mutation is first presented by Fogel in 1988 [FOG 88] and further discussed by Michalewicz in 1992 [MIC 92]. It operates as follows. First, it randomly selects a position in the string representing the individual to mutate and removes the corresponding symbol. In the second step, the other symbols are shifted to fill the empty position. The last step consists of selecting another random position in the string to insert the previously removed symbol (Figure 4.11(a)).

4.3.2.2. *Mutation by inversion (IVM)*

The inversion mutation operator was introduced by Holland [HOL 75]. Later it was revisited by David Fogel [FOG 88] for application to the TSP problem. The first implementation of this operator is known as *simple inversion mutation*. Fogel improved this operator such that it allows the

Genetic Algorithm to preserve the feasibility of the evolved solutions without using a repair method. He then proposed the following procedure.

First, the operator selects two random positions on the parent string. Then, the sub-strings between these two positions are inverted. The other symbols are copied to the offspring without any changes. This operator preserves most adjacency information and only breaks two edges, but it leads to the disruption of order information (Figure 4.11(b)).

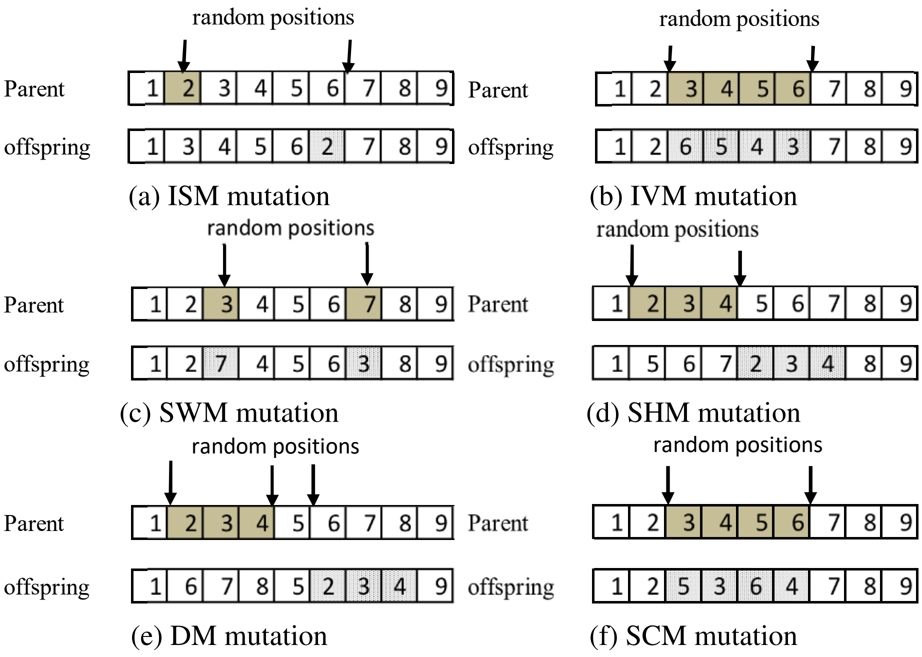


Figure 4.11. *Example of Order-based mutation operators*

4.3.2.3. *Mutation by exchange or swap mutation (SWM)*

The swap mutation is a very simple operator proposed by Oliver *et al.* in 1987 [OLI 87]. SWP begins by selecting two random alleles along the parent string. Next, the corresponding symbols are exchanged as illustrated in Figure 4.11(c). SWM produces very small perturbations if the parent has a large size (e.g. TSP with hundreds of cities), which do not help the algorithm to avoid premature convergence when parents become similar late in the evolution. To counter this problem, Ambati proposed the *repeated exchange* *operator* [AMB 91]. Byer also proposed an extension of SWM to a three-point-exchange mutation [BEY 92] that swaps three randomly chosen points in a genotype and changes up to six edges per mutation.

4.3.2.4. *Mutation by shifting (SHM)*

First, shift mutation chooses randomly two positions on the parent. Then, the symbol in the first position is shifted from its current position to the second position. Illustration of the shift mutation is shown in Figure 4.11(d).

4.3.2.5. *Displacement Mutation (DM)*

Displacement mutation is defined as the ISM (section 4.3.2.1) [MIC 92]; however, instead of displacing a symbol in the parent genotype as ISM, it displaces a sub-sequence. It operates as follows.

DM first selects randomly a sub-sequence of symbols, removes this sub-sequence from the genotype and re-inserts it in another random place (Figure 4.11(e)).

4.3.2.6. *Scramble Mutation (SCM)*

The scramble mutation operator [SYS 91] selects a random sub-sequence in the parent genotype and scrambles the symbols in it (Figure 4.11(f)).

4.3.3. *Specific operators*

Many other operators were proposed for the evolutionary combinatorial optimization that are specific to the problem to be solved. Researchers introduced application-dependent variation operators when generic operators are not applicable or have poor performance for the considered problem. For example, for the job machine ordering problem, Kobayashi *et al.* proposed the Subsequence Exchange Crossover that uses a special encoding process [ONO 97]. They also proposed the Job-based order crossover that is a variant of the first operator. For scheduling problems, Gen *et al.* proposed the Partial schedule Exchange Crossover for job-shop scheduling problems [GEN 94]. It is designed with the operation-based encoding that is based on the idea of building blocks introduced by Holland. For the same class of problems, Cheng, Tsujimura and Gen proposed the Neighborhood Search-based Mutation that replaces a schedule by its λ-optimum neighborhood, where λ is the number of symbols permuted [CHE 99]. As additional examples of specific operators, we can cite: the Product Geometric Crossover for Sudoku problem [MOR 06], the Generalized Cycle crossover and Labeling-independent Crossover for Graph Partitioning Problems [MOR 07]and the Quotient Geometric Crossovers for grouping, graph, sequence and glued space applications [YOO 12].

Listing the application dependent variation operators might be quite lengthy regarding the great number of applications. Further examples can be found in [GEN 00] and [ONW 09].

4.3.4. *Discussion*

Given the large number of crossover and mutation operators proposed for the evolutionary combinatorial optimization, choosing some operators for solving a given problem becomes in itself an optimization problem. Several studies have been published comparing the efficiency of a set of variation operators. We summarize in this section some analyses from the literature and discuss the conclusions given by authors.

Initially, the crossover operators developed for the combinatorial evolutionary optimization serve two main purposes:

1) producing valid (feasible) solution in order to avoid the use of repair methods or penalization techniques;

2) producing offspring that preserve some sequences from the parents in order that offspring inherit useful information from the best parents.

However, evolving a population using only order crossover might increase the computational cost. Otherwise, only applying swaps and inversion between symbols with mutation operators does not help to escape local optima. Several authors tried to give some guidelines in choosing variation operators by proposing a special ranking defined with a set of comparative tests. Published experimental studies show that the behavior and the relative ranking of the operators depend on the problem.

Besides ranking, other works proposed some extensions to enhance the exploration and/or exploitation capacity of the operators. For example, several extensions were proposed for the order-based crossover (OX). Deep and

Mebrahtu [DEE 11] compared five variations of OX and demonstrated, by including some modifications in the choice of the cut points, that OX is able to perform better in solving some TSP benchmark with a number of cities greater than 100. However, it is still expensive, in terms of computational time, for solving problems with a size lower than 100.

Puljic *et al.* compared eight crossover operators for the vehicle routing problem, which deals with the scheduling of a fleet of vehicles to distribute goods between a depot and customers [PUL 13]. The implemented operators are OX, PMX, ERX, CX, AEX and Heuristic crossover (HGreX, HRndX, HProX). The tests were done on seven benchmark VRP instances. The authors concluded that the performance of the heuristic crossover and AEX is usually better than that of the other operators, especially PMX and CX which show very low performance. They also noted that mutation always improves the general performance.

The poor efficiency of PMX and CX is also reported in other studies. A well-known comparative study with an extended review of operators proposed for the TSP problem was published by Larrangua *et al.* [LAR 99] in 1999. In this study, authors implemented eight crossover operators and six mutation operators. They tried 48 operator combinations on three different TSP benchmark tests. They noted that the operator ranking is problem-dependent as we mentioned above; however, a certain uniformity of behavior of the operators in the different examples can be seen. For example, the crossover operators ERX, OX1 (initial version of OX) and POS as well as the mutation operators DM, IVM and ISM provided the best results.

ERX and OX were also in the top ranking in the study published by Starkweather *et al.* [STA 91]. This study is an empirical comparison of six crossover operators, ERX, OX1, OX2, POS, PMX and CX, in solving an instance of the TSP problem with 30 cities without the help of mutation operators.

Having observed the highly variable performance of crossover operators, whether blind or heuristic, and in order to reduce the general dependence of GEA on the implemented variation operators, some authors proposed the use of multi-crossover mechanism combining several reproduction and mutation strategies. Puljic *et al.* [PUL 13] proposed a mix of eight crossovers, where any of them is activated randomly with equal probability. Similarly, a mix of three mutations is implemented. The set of operators is the same developed for the comparative study described above. Through the analysis of the obtained results, they concluded the following fact:

> *Mixing different crossover operators produces better results than using any operator alone. Thus, synergy among crossovers seems to exist. Improvement is more visible when there is no mutation.*

Mixing operators has also been recently proposed by Osaba *et al.* [OSA 14] in their Adaptive Multi-Crossover Population Algorithm (AMCPA). AMCPA uses different reproduction functions alternatively with a dynamic crossover rate adapted according to the search performance and the current generation number. AMCPA was applied on six combinatorial problems: symmetric and asymmetric TSP, Capacitated VRP, VRP with Backhauts, N-Queens and one-dimensional Bin Packing. The experimental study showed that the AMCPA outperforms the GEAs in terms of solution quality and run times in all the problems.

Note that operator mixing enhances the exploration capacity of the Genetic Algorithm; however, it could reduce the convergence speed. Therefore, it is necessary to control the crossover rate along the evolution.

According to the statistical analysis given in the majority of the published comparative studies, the following general observations can be stated for crossover operators:

1) adjacency-based and heuristic crossover in general perform better than position-based crossover;

2) some classic crossovers (such as the CX or the MPX) are not able to deal with very large combinatorial problems;

3) the reproduction step reduces the speed of the GEA.

For these reasons, great efforts have been made to improve the reproduction operators when solving the combinatorial problem. Through the state of the art, it can be noted that early efforts to enhance the GEA efficiency on solving combinatorial optimization problems were oriented basically to the improvement of the reproduction mechanism by proposing new crossover operators or mixing the existing operators. Later, some studies have demonstrated that the efficiency of crossover operators depends on the problem and the parameter settings. Then, authors proposed, instead of varying operators, to control the GEA parameters and introduce to the algorithm a mechanism that allows it to dynamically adjust its mutation and crossover rates. Recently, some researchers tried to demonstrate, through experimental studies and statistical analysis, that the reproduction step is not necessary to solve a combinatorial problem efficiently with GEA. For example, Osaba *et al.* [OSA 13b, OSA 13a] came up with the following hypothesis:

> *The crossover phase of the genetic algorithms is not efficient for the search process and the capacity of optimization of the technique when it is applied to routing problems using path encoding.*

The authors supposed that the crossover operators offered no advantage in the optimization process, and that the most important steps were selection and mutation. To prove this hypothesis, they made several series of experiments with two different GEAs:

1) Traditional GEAs using either blind crossover, which only takes care of generating individuals that meet the constraints (CX, OX, MOX, OBX), or heuristic crossover that make small optimization on the resulting individuals;

2) GEAs using only mutation and survivor selection.

According to the statistical analysis of the obtained results [OSA 13b, OSA 13a], the authors made the following conclusions:

1) the use of CX operator does not give any improvement to the process of optimizing a GEA;

2) the crossover phase increases the complexity of the GEA without providing any visible improvement.

As a final conclusion, the authors proposed to use only mutation functions and survival mechanisms to solve combinatorial problems with GEA or to extend the GA with a local search.

The idea of combining GEA with a local search for combinatorial problems was proposed and supported by several researchers many years ago. They believed that implementing a local search was crucial for the efficiency of any evolutionary algorithm. The crossover operators are not removed from the evolutionary system but applied to locally optimal individuals, i.e. individuals improved by a local search. Muhlunbein implemented a parallel

Genetic Algorithm, where both parents and offspring are locally optimized by applying hill-climbing [MÜH 91]. Parents are locally improved before crossing-over, and offspring are improved before replacement. The search space is explored mainly by crossover. A diversification mechanism could be added with the Islands model. The algorithm was successfully experimented on complex instances of TSP.

Since the 1990s, this kind of hybridization, known as *Memetic Algorithm*, has been well studied. This is an evolutionary algorithm, where all solutions are locally optimized before evaluation. The "Handbook of Memetic Algorithms" [NER 12] describes many contributions and applications using memetic Algorithm. It shows how this approach could be highly effective in practice.

4.4. Conclusion

This chapter has presented the main generic variation operators dedicated to Binary and Order-based representations, for solving with GEAs the most studied combinatorial optimization problems in the literature.

The majority of the variations operators have been introduced primarily for solving the TSP, which resulted in that Evolutionary Computation research has been criticized for considering artificial test problems that are far from or much simpler than the real-life cases. Researchers in the domain have reacted to this criticism by considering realistic problems. They then proposed several extensions for the generic operators and introduced some specific operators that are problem-dependent. They have also worked to improve the GEA by introducing adaptive or parallel implementations. However, the robustness of evolutionary algorithms is greatly enhanced when they are hybridized with other optimization methods like local search techniques. The number of papers introducing hybrid systems is growing, indicating that there is a trend towards this direction. As further reading in this field, we might suggest the following two books: "Differential evolution: a handbook for global permutation-based combinatorial optimization" [ONW 09] and "Handbook of memetic algorithms" [NER 12].

5 Multi-objective Optimization

5.1. Introduction

Decisions regarding real-world problems such as the design of products, organizations, or processes, often depend on compromises, resulting from dialogues and negotiations between several parties. This context makes it difficult or even impossible to fully formalize such problems and to solve them completely by means of computer systems. For instance, how should we decide between performance maximization, security maximization, environmental nuisance minimization and cost minimization when designing a new machine? In general, no global objective function is known to best take into account such contradictory criteria.

Multi-objective or multi-criteria optimization consists of simultaneously optimizing several objectives or criteria, generally leading to compromises being made between them. Such problems have multiple (possibly infinite) incomparable alternative solutions. Decision makers often need representative sets of alternatives to select the best ones according to their visions of the problems they have to solve. In this way, multi-objective optimization has to build a representative sample of the diversity of the best alternatives.

Let $\mathbf{f}(\mathbf{x})$ be a vector of c objectives associated with an instance of solution $\mathbf{x}$ of a multi-objective optimization problem. Each of its components $f_i(\mathbf{x})$ is equal to the value of the i^{th} objective for solution $\mathbf{x}$. Without loss of generality, we will consider in the following sections the case in which all the objectives of a problem have to be minimized. Indeed, when a problem does not satisfy this condition, it is enough to change the signs of the objectives which must be maximized.

5.2. Problem formalization

5.2.1. *Pareto dominance*

Let us consider two vectors of objectives $\mathbf{v}$ and $\mathbf{u}$. If all the components of $\mathbf{v}$ are less than or equal to the components of $\mathbf{u}$, with at least one strictly lower component, then the vector $\mathbf{v}$ corresponds to a better solution than $\mathbf{u}$. In this case, $\mathbf{v}$ *dominates* $\mathbf{u}$ in the Pareto sense. In a more formal way, it can be written as: $\mathbf{v} <_P \mathbf{u}$.

$$\mathbf{v} <_P \mathbf{u} \Longleftrightarrow \forall i \in \{1, ..., c\}, v_i \leq u_i \text{ and } (\exists j \in \{1, ..., c\}, v_j < u_j)$$

Figure 5.1 represents the dominance relations between four objective vectors in a two-dimensional space. c is dominated by a and b. d is only dominated by b. a and b are not dominated.

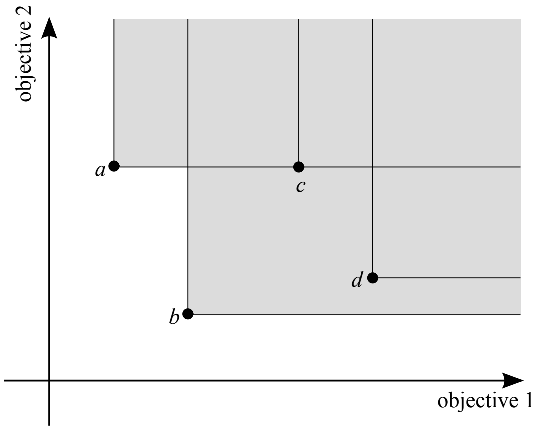


Figure 5.1. *Dominations in the Pareto sense in an objective space of dimension 2. The gray area represents the dominated objective vector set*

It is also said that $\mathbf{y} \in \mathbf{\Omega}$ dominates $\mathbf{x} \in \mathbf{\Omega}$, which is denoted as $\mathbf{y} \prec \mathbf{x}$ if and only if $\mathbf{f}(\mathbf{y}) <_P \mathbf{f}(\mathbf{x})$ for a minimization problem.

5.2.2. *Pareto optimum*

The set of the objective vectors that cannot be dominated constitutes the optimal values of the problem in the Pareto sense. This means that if a component of such an objective vector is improved, another component at least deteriorates. These vectors belong to the *Pareto front*, or *trade-off surface*, denoted as $\mathcal{P}$:

$$\mathcal{P} = \{\mathbf{f}(\mathbf{x})|\mathbf{x} \in \mathbf{\Omega},\ \nexists \mathbf{y} \in \mathbf{\Omega}, \mathbf{y} \prec \mathbf{x}\}$$

The *Pareto-optimal set* $\mathbf{X}^*$ is defined as the set of solutions in search space $\mathbf{\Omega}$ whose objective vectors belong to the Pareto front:

$$\mathbf{X}^* = \{\mathbf{x} \in \mathbf{\Omega}|\mathbf{f}(\mathbf{x}) \in \mathcal{P}\}$$

5.2.3. *Multi-objective optimization algorithms*

The multi-objective optimization consists of building the Pareto optimal set $\mathbf{X}^*$. However, $\mathbf{X}^*$ can contain an infinite number of solutions if the search space is continuous. Even if $\mathbf{\Omega}$ is finite, a decision maker will be able to exploit $\mathbf{X}^*$ effectively if it is not too large. Thus, it is expected that the multi-objective optimization algorithm (MOEA) can produce a set or a subset of non-dominated solutions, not too large, such that they are as close as possible to the Pareto front, by *covering it as evenly and as completely as possible* [DEB 01].

5.3. The quality indicators

There is a wide choice of multi-objective optimization algorithms, each with its own characteristics. These algorithms often use parameters whose values can strongly influence the quality of results, while their values are hard to estimate to best achieve the goals of the user. Too often, the practitioners will have no other way to select the best approach than comparing the results provided by several algorithms and parameter sets. It is therefore important that they have quality indicators to facilitate performance analysis of the tested approaches.

Many quality indicators have been proposed for multi-objective optimization [KNO 02, ZIT 03]. One of them is described below because of its good properties, although it requires high computational power.

5.3.1. *The measure of the hypervolume or "$\mathcal{S}$-metric"*

Let $\boldsymbol{\rho} = (\rho_1, ..., \rho_c)$ be a reference point in the objective space and let $\mathbf{a} = (a_1, ..., a_c)$ be an element of a set $\mathbf{A}$ of non-dominated objective vectors. $a_i \leq \rho_i$ is required when minimizing the objectives. $\boldsymbol{\rho}$ and $\mathbf{a}$ define a hyper-rectangle whose edges are parallel to the coordinate axes of the objective space. The expression of its hypervolume is:

$$v(\mathbf{a}, \boldsymbol{\rho}) = \prod_{i=1}^{c} (\rho_i - a_i)$$

Set $\mathbf{A}$ and point $\boldsymbol{\rho}$ define a hypervolume $v(\mathbf{A}, \boldsymbol{\rho})$ in the objective space by the union of the hyper-rectangles associated with elements of $\mathbf{A}$ (Figure 5.2). $\boldsymbol{\rho}$ is chosen so that each of its coordinates is an upper bound of the coordinates of all points of $\mathbf{A}$ (for a minimization of the objectives). The measure of hypervolume $v(\mathbf{A}, \boldsymbol{\rho})$ is a good comparison indicator of non-dominated sets because it is strictly monotonic according to the Pareto dominance relation [KNO 02]. Namely, if any element of a set $\mathbf{B}$ is dominated by at least one element of $\mathbf{A}$, then hypervolume $v(\mathbf{B}, \boldsymbol{\rho})$ is less than hypervolume $v(\mathbf{A}, \boldsymbol{\rho})$. So far, indicator $v(\mathbf{A}, \boldsymbol{\rho})$ is the only one that has this monotony property, which explains the interest being shown in it.

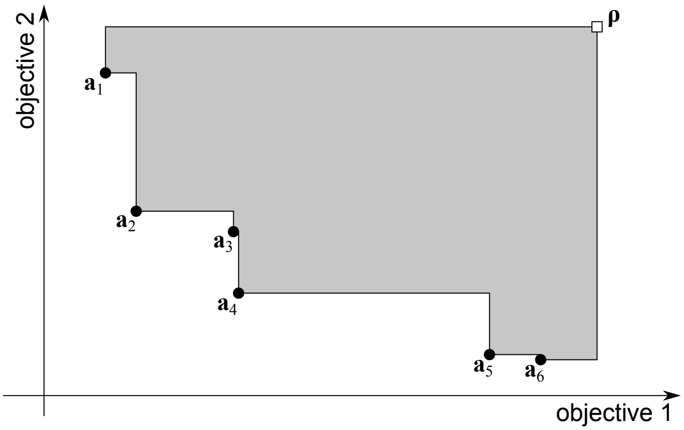


Figure 5.2. *The hypervolume for a two-objective minimization problem $v(\mathbf{A}, \boldsymbol{\rho})$ with $\mathbf{A} = \{\mathbf{a}_1, ..., \mathbf{a}_6\}$ is given by the gray area*

The maximum hypervolume is obtained when $\mathbf{A}$ is the Pareto front. A value $v(\mathbf{A}, \boldsymbol{\rho})$ near the maximum indicates that the non-dominated vectors of a set $\mathbf{A}$ are close to the Pareto front, with a good coverage quality.

The main drawback of this metric lies in its exponential computational overhead in the number of objectives. The reference point $\boldsymbol{\rho}$ must also be appropriately chosen. There are several approaches for calculating hypervolumes. [WHI 12] is one of the recent references in this field.

5.4. Multi-objective evolutionary algorithms

Evolutionary algorithms are well suited for simultaneous searching of a collection of optimal solutions because they deal with populations of solution instances. The evolutionary approach requires *a priori* the implementation of an archive of the non-dominated solutions discovered during a complete evolution. Indeed, there is no guarantee that at the end of the evolution, the solutions that have best approached the Pareto optimal set have been preserved in the population. Thus, at the end of each generation, the population is copied in the archive, then the dominated individuals are removed from it. However, the management of an external archive could be useless for multi-objective optimization algorithms that implement a form of elitism.

Two kinds of evolutionary approaches are widely considered in the literature:

– the methods using a *Pareto ranking* to evaluate the fitness function;

– the *aggregation* methods that transform a multi-objective optimization problem into a collection of single-objective problems. The exact (or approximate) resolution of each single-objective problem then gives a point on (or close to) the Pareto front.

5.5. Methods using a "Pareto ranking"

These methods were the first ones to show their efficiency in producing a good coverage of a Pareto front. The individuals of a population correspond to instances of solutions in the search space. An objective vector is evaluated and assigned to each of them. Then, each individual gets a scalar fitness value, computed with the objective vectors, such that the non-dominated individuals will be selected more often than the others.

The even coverage of the Pareto front, or at least, its nearest non-dominated solution set found, is obtained by a diversity preservation mechanism within the population.

There is a difficulty in applying the techniques based on Pareto dominance, related to the dimensionality of the objective space. The more objectives to optimize, the vaster the Pareto front is, and the less likely individuals are dominated by others. In this case, if a maximum fitness is assigned to the non-dominated individuals, in order to favor their reproduction, then many individuals will have this fitness. This situation generates a low selection pressure, and consequently a (very) slow convergence of the algorithm. Thus, the strategies using the Pareto dominance have to take this problem into account. Currently, the "Pareto ranking" approach makes it difficult to go beyond problems involving four objectives.

A first approach of "Pareto ranking" was proposed by D.E. Goldberg in his famous book [GOL 89]. However, he did not describe any concrete implementation of this algorithm, and obviously, he did not provide any performance result. The idea, however, inspired many researchers in the following years. It has given birth to the first generation of multi-objective methods using a Pareto ranking, such as MOGA (*Multiple Objectives Genetic Algorithm*) [FON 93], NPGA (*Niched Pareto Genetic Algorithm*) [HOR 94] and NSGA (*Non-Dominated Sorting Genetic Algorithm*) [SRI 94].

In the 2000s, these approaches were improved by the introduction of elitism, which gave birth to the second generation of multi-objective methods. Among the most widely used algorithms are NSGA-II [DEB 02], SPEA (*Strength Pareto Evolutionary Algorithm*) [ZIT 99], SPEA2 [ZIT 02], PAES (*Pareto Archived Evolution Strategy*) [KNO 00], MOMGA *(Multi-Objective Messy Genetic Algorithm)* [VEL 00] and its extension MOMGA-II [ZYD 01]. The widespread and efficient NSGA-II is described below.

5.5.1. *NSGA-II*

The NSGA-II method [DEB 02] was introduced in 2002 as an improvement of NSGA on the following points:

– algorithmic complexity reduced to $\mathcal{O}(\mu^2 c)$;

– diversity preservation mechanism (niching) without parameter;

– implementation of elitism (section 1.3.6.6, p. 19) to accelerate the convergence of the algorithm.

Algorithmic complexity of the Pareto ranking

The Pareto ranking of NSGA-II is composed of an initialization phase followed by the rank assignment phase. During the initialization phase described by the algorithm 1, the following are associated with each individual i of population $\mathbf{P}$:

– a domination counter α_i giving the number of individuals that dominate i;

– the set of individuals $\mathbf{S}_i$ dominated by i.

```
$\mathcal{F}_1 \leftarrow \emptyset$
**for each** *individual* $i \in \mathbf{P}$ **do**
    $\mathbf{S}_i \leftarrow \emptyset$
    $\alpha_i \leftarrow 0$
    **for each** *individual* $j \in \mathbf{P}$ **do**
        **if** $i \prec j$ **then**
            $\mathbf{S}_i \leftarrow \mathbf{S}_i \cup \{j\}$
        **end**
        **else if** $j \prec i$ **then**
            $\alpha_i \leftarrow \alpha_i + 1$
        **end**
    **end**
    **if** $\alpha_i = 0$ **then**
        $\mathcal{F}_1 \leftarrow \mathcal{F}_1 \cup \{i\}$
        $r_i \leftarrow 1$ ;                    // $r_i$:  non-domination rank of $i$
    **end**
**end**
```

Algorithm 1: Pareto ranking of NSGA-II: initialization

The individuals for which α_i is zero constitute the set of all the non-dominated individuals of rank 1, denoted as $\mathcal{F}_1$. The constructions of $\mathbf{S}_i$ and computations of α_i for all individuals require $\mu^2 c$ comparisons.

The rank assignment phase (algorithm 2) for all the individuals of the population follows the initialization phase. Assuming that set $\mathcal{F}_r$ of the rank r non-dominated individuals was built, it is possible to determine the rank $r+1$ non-dominated individuals as follows: for any individual i belonging to $\mathcal{F}_r$, counters α_j of the individuals j dominated by i are decremented. Individuals j for which $\alpha_j = 0$ constitute the set $\mathcal{F}_{r+1}$. The complexity of this algorithm is also $\mathcal{O}(\mu^2 c)$. The fitness value of an individual is given by its rank that the evolutionary algorithm tends to minimize.

```
$r \leftarrow 1$
**while** $\mathcal{F}_r \neq \emptyset$ **do**
    $\mathcal{F}_{r+1} \leftarrow \emptyset$
    **for each** *individual* $i \in \mathcal{F}_r$ **do**
        **for each** *individual* $j \in \mathbf{S}_i$ **do**
            $\alpha_j \leftarrow \alpha_j - 1$
            **if** $\alpha_j = 0$ **then**
                $\mathcal{F}_{r+1} \leftarrow \mathcal{F}_{r+1} \cup \{j\}$
                $r_j \leftarrow r + 1$
            **end**
        **end**
    **end**
    $r \leftarrow r + 1$
**end**
```

Algorithm 2: Pareto ranking of NSGA-II: rank assignment

Diversity preservation mechanism

This mechanism uses a binary tournament selection (section 1.3.4, p. 16) specific to NSGA-II referred to as the "*crowded tournament*". This tournament is designed to favor the selection of individuals with the same non-domination rank in sparsely populated areas in either the objective space or the search space Ω, depending on the user choice. The explanations in the following lines are related to the objective space. The adaptation to the search space is direct.

The crowded tournament is based on a *crowded-comparison* operator denoted as $\prec_n$. The "*crowding distance*" d_i is associated with each individual i. It represents an estimate of the distance between i and its neighbors in the space of the objectives. Let r_i be the non-domination rank of individual i. The crowded comparison operator is defined below:

$$i \prec_n j \Longleftrightarrow r_i < r_j \text{ or } (r_i = r_j \text{ and } d_i > d_j)$$

The crowded tournament between two individuals i and j selects i if $i \prec_n j$.

The designers of the method propose calculating the crowding distance d_i as follows. Let $f_m(i)$ be the value of objective m for individual i of $\mathcal{F}_r$ with a given value for r, we define:

– $f_m^{\max}$: maximum of objective m in the population;

– $f_m^{\min}$: minimum of objective m in the population;

– $f_m^+(i)$: closest value to $f_m(i)$ in $\mathcal{F}_r$ such that $f_m^+(i) \geq f_m(i)$. For extreme individuals i with $f_m(i)$ maximum in $\mathcal{F}_r$, $f_m^+(i)$ is set equal to ∞ for one of them and $f_m^+(i) = f_m(i)$ for the others, if any. This is useful in order for extreme individuals to be selected with a sufficiently high probability to explore their neighborhoods with the variation operators (mutation and crossover);

– $f_m^-(i)$: closest value to $f_m(i)$ in $\mathcal{F}_r$ such that $f_m^-(i) \leq f_m(i)$. For extreme individuals i with $f_m(i)$ minimum in $\mathcal{F}_r$, $f_m^-(i)$ is set equal to $-\infty$ for one of them and $f_m^-(i) = f_m(i)$ for the others, if any.

The crowding distance d_i is expressed as:

$$d_i = \sum_{m=1}^{c} \frac{f_m^+(i) - f_m^-(i)}{f_m^{\max} - f_m^{\min}}$$

Figure 5.3 shows an example of calculation of the crowding distance for an individual i in a two-dimensional space of objectives.

Algorithm 3 describes the computation of d_i for sub-population $\mathcal{F}_r$ of non-dominated individuals i with rank r. It allows the computational complexity of the crowding distance calculation to be obtained, which is $\mathcal{O}(c\mu \log \mu)$. This calculation follows the rank assignment to each individual with complexity equal to $\mathcal{O}(\mu^2 c)$. Consequently, the overall complexity of these two operations is $\mathcal{O}(\mu^2 c)$.

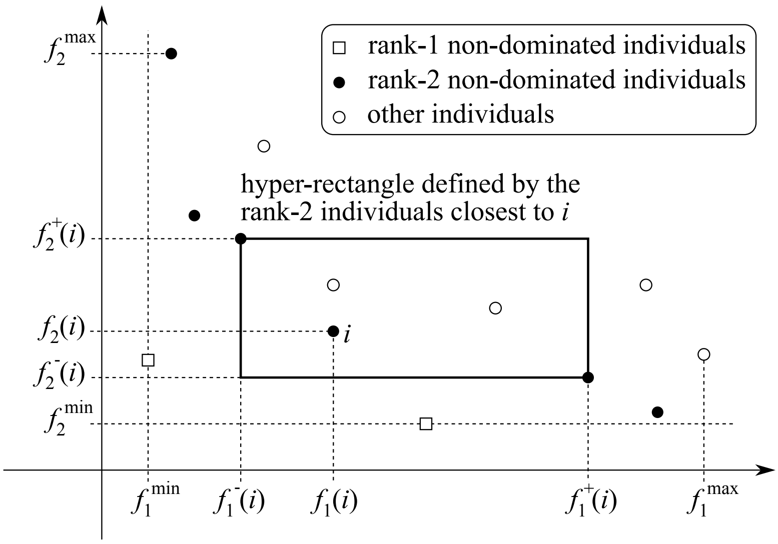


Figure 5.3. *Calculation of the crowding distance in a two-dimensional space of objectives for individual i of rank 2:*

$$d_i = \frac{f_1^{+}(i) - f_1^{-}(i)}{f_1^{\max} - f_1^{\min}} + \frac{f_2^{+}(i) - f_2^{-}(i)}{f_2^{\max} - f_2^{\min}}$$

```
$l \leftarrow |\mathcal{F}_r|$        // $l$ is the number of non-dominated individuals of rank $r$
**for** *each individual* $i \in \mathcal{F}_r$ **do**
    $d_i \leftarrow 0$
**end**
**for** *each objective* $m$ **do**
    $\mathbf{T} \leftarrow \text{sort}(\mathcal{F}_r, m)$
     // $\mathbf{T}$ is an array of individuals of $\mathcal{F}_r$ sorted according to objective $m$
    $d_{\mathbf{T}[1]} \leftarrow d_{\mathbf{T}[l]} \leftarrow \infty$
    **for** $k = 2 \;\; l - 1$ **do**
        $d_{\mathbf{T}[k]} \leftarrow d_{\mathbf{T}[k]} + (f_m(\mathbf{T}[k+1]) - f_m(\mathbf{T}[k-1]))/(f_m^{\max} - f_m^{\min})$
    **end**
**end**
```

Algorithm 3: Computation of crowding distances d_i for any individual i of $\mathcal{F}_r$

Elitism

For a generation $g > 0$, the new generation is obtained by creating a population of children $\mathbf{Q}_g$ from population $\mathbf{P}_g$ by applying in sequence the crowded tournament selection, crossover and mutation operators (Figure 5.4). The size of $\mathbf{Q}_g$ is chosen by the authors equal to the one of $\mathbf{P}_g$, i.e. μ. The Pareto ranking described above is applied to the union of $\mathbf{Q}_g$ and $\mathbf{P}_g$, which makes it possible to compute the non-domination ranks r_i of the individuals and to generate sub-populations $\mathcal{F}_r$. Parents and children participate in the same ranking, which implements elitism.

Environmental selection operator

The population $\mathbf{P}_{g+1}$, built by the environmental selection operator, is first composed of individuals of sub-populations $\mathcal{F}_1$ to $\mathcal{F}_k$, where k is the greatest integer such that the sum of the sizes of these sub-populations is less than or equal to μ. To complete $\mathbf{P}_{g+1}$ up to μ individuals, those of $\mathcal{F}_{k+1}$ are sorted with comparison operator $\prec_n$ and the best solutions are inserted into population $\mathbf{P}_{g+1}$ until it contains μ individuals.

The initial population

The initial population $\mathbf{P}_0$ is generated with a problem-dependent method if available or else by construction of random individuals. Pareto ranking is then applied to $\mathbf{P}_0$ to calculate the initial fitness values of its individuals. This is different for the other generations for which this ranking is applied to the union of $\mathbf{P}_g$ and $\mathbf{Q}_g$.

The generational loop

Figure 5.4 depicts the generational loop of NSGA-II. The two steps of the calculation of composite fitnesses (r_i, d_i) for individual i are highlighted.

In conclusion

The NSGA-II method is recognized as being highly effective. Today, it is one of the reference methods for multi-objective evolutionary optimization when the number of objectives is less than four.

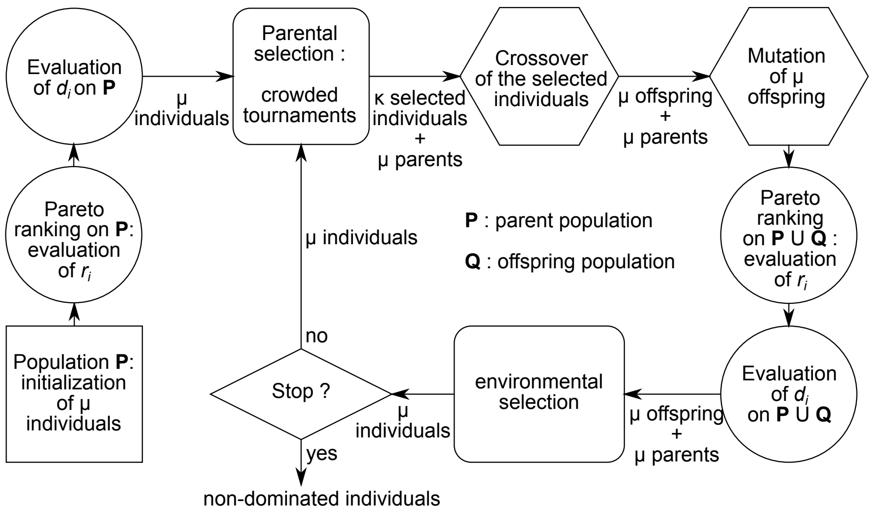

Evaluation of d_i on **P**
μ individuals
Parental selection : crowded tournaments
κ selected individuals + μ parents
Crossover of the selected individuals
μ offspring + μ parents
Mutation of μ offspring
Pareto ranking on **P** U **Q** : evaluation of r_i
Evaluation of d_i on **P** U **Q**
μ offspring + μ parents
environmental selection
μ individuals
Stop ?
yes
no
non-dominated individuals
μ individuals
P : parent population
Q : offspring population
Pareto ranking on **P**: evaluation of r_i
Population **P**: initialization of μ individuals

Figure 5.4. *The generational loop of NSGA-II.*

5.6. Many-objective problems

As previously explained in section 5.5, Pareto dominance-based methods are efficient when problems have no more than three objectives. Furthermore, the size of the Pareto front increases exponentially with the number of objectives: for two objectives, the Pareto front can be a line; for c objectives, it can be a $(c-1)$-dimensional manifold. Thus, preserving diversity when faced with many objectives, in such a way that the Pareto front is evenly covered by the individual of a population, is difficult to achieve. Authors have proposed methods to solve problems with many objectives. They belong to the following categories, which are presented in a survey published by Bingdong Li *et al.* [LI 15]:

– relaxed dominance approaches;

– aggregation-based approaches;

– indicator-based approaches;

– diversity-based approaches;

– reference set approaches;

– preference-based approaches;

– dimensionality reduction approaches.

5.6.1. *The relaxed dominance approaches*

This class of methods substitutes the Pareto dominance with another dominance rule, which increases the likelihood that a solution $\mathbf{x}$ dominates another solution $\mathbf{y}$. For instance, ε-dominance relaxes the Pareto dominance such that for a maximization problem, $\mathbf{x}$ "ε-dominates" $\mathbf{y}$, denoted as $\mathbf{x} \succ_\varepsilon \mathbf{y}$ if and only if:

$$\forall i \in \{1, ..., c\}, (1+\varepsilon) f_i(\mathbf{x}) \geq f_i(\mathbf{y})$$

where f_i is the i-th objective function and $\varepsilon > 0$ is a parameter of the method [LAU 02]. Obviously, $\mathbf{x} \succ \mathbf{y} \Rightarrow \mathbf{x} \succ_\varepsilon \mathbf{y}$. The efficient ε-MOEA algorithm uses a form of ε-dominance combined with an aggregation of the objectives related to a set of reference points placed on the nodes of a grid [DEB 03].

Other relaxed dominance approaches have been proposed as the "$(1-k)$-dominance" [FAR 02]. Let c be the number of objectives, let n_b be the number of objectives such that $f_i(\mathbf{x})$ is better than $f_i(\mathbf{y})$, $i \in \{1, ..., c\}$ and let n_e be the number of objectives such that $f_i(\mathbf{x}) = f_i(\mathbf{y})$. $\mathbf{x}$ $(1-k)$-*dominates* $\mathbf{y}$ if and only if:

$$\begin{cases} n_e < c \\ n_b \geq \frac{c-n_e}{k+1} \end{cases}$$

where $k \in [0, 1]$ is a user parameter. If $k = 0$, the "$(1-k)$-dominance" becomes the Pareto dominance.

5.6.2. *Aggregation-based approaches*

An *aggregation function* of the objectives transforms a multi-objective problem into a mono-objective one. It should be chosen in such a way that its optimum corresponds to one point of the Pareto front. Aggregation functions $G_p(\mathbf{x}|\mathbf{w}, \boldsymbol{\rho})$ defined below are often used:

$$G_p(\mathbf{x}|\mathbf{w}, \boldsymbol{\rho}) = \sqrt[p]{\sum_{i=1}^{c} (w_i |f_i(\mathbf{x}) - \rho_i|)^p} \qquad [5.1]$$

$G_p(\mathbf{x}|\mathbf{w}, \boldsymbol{\rho})$ is a weighted distance between the objective functions $\mathbf{f}(\mathbf{x})$ and a reference point $\boldsymbol{\rho}$. $\boldsymbol{\rho}$ is chosen such that $\forall i \in \{1, ..., c\}$, $\forall \mathbf{x} \in \boldsymbol{\Omega}$, $\rho_i \leq f_i(\mathbf{x})$, for a minimization problem. For each weight vector $\mathbf{w} = (w_i)$ with $w_i > 0$ and $\sum_{i=1}^{c} w_i = 1$, there exists at least one Pareto optimal solution. Each weight reflects the importance given to its associated objective by a decision maker.

The most common values for p are:

– $p = 1$: Manhattan distance;

– $p = 2$: Euclidean distance;

– $p = \infty$: Chebyshev distance: $G_\infty(\mathbf{x}|\mathbf{w}, \boldsymbol{\rho}) = \max_i (w_i|f_i(\mathbf{x}) - \rho_i|)$.

If $p = 1$, the minimization of $G_1(\mathbf{x}|\mathbf{w}, \boldsymbol{\rho})$ is in fact the minimization of $\sum_{i=1}^{c} w_i f_i(\mathbf{x})$ for a minimization problem because $\boldsymbol{\rho}$ is constant. This choice for p is attractive due to its simplicity. However, minimizing $G_1(\mathbf{x}|\mathbf{w}, \boldsymbol{\rho})$ cannot reach all the points of a Pareto front when it includes concave parts, regardless of the values of $\mathbf{w}$, as shown in Figure 5.5.

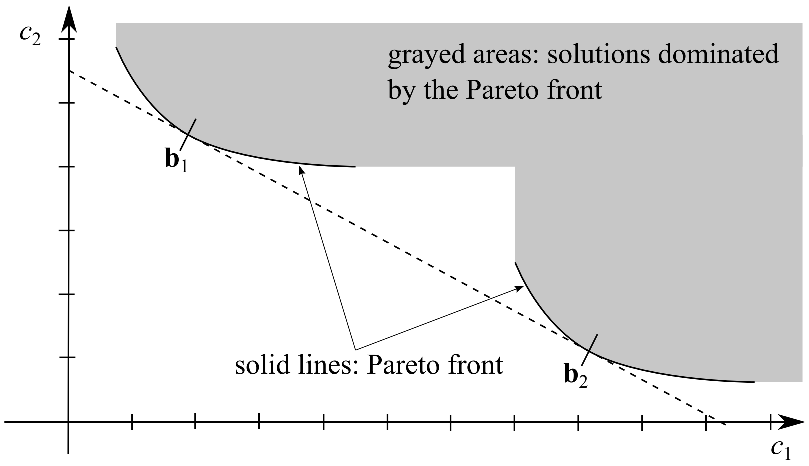

c_2
grayed areas: solutions dominated by the Pareto front
$\mathbf{b}_1$
solid lines: Pareto front
$\mathbf{b}_2$
c_1

Figure 5.5. *Aggregation of the objectives with the weighted-sum method: the points of the Pareto front between $\mathbf{b}_1$ and $\mathbf{b}_2$ cannot minimize $G_1(\mathbf{x}|\mathbf{w})$ for all $\mathbf{w}$*

Conversely, choosing $p = \infty$ allows the minimum of $G_\infty(\mathbf{x}|\mathbf{w}, \boldsymbol{\rho})$ to reach all points of a Pareto front according to the values of $\mathbf{w}$. However, dominated solutions can also give minimal values for $G_\infty(\mathbf{x}|\mathbf{w}, \boldsymbol{\rho})$. Thus, Figure 5.6 shows the case where the objective vectors on the line segment $(\mathbf{b}_1, \mathbf{b}_2)$, except $\mathbf{b}_1$, are dominated by $\mathbf{b}_1$, although they give the minimal value 2 for function $G_\infty(\mathbf{x}|\mathbf{w}, \boldsymbol{\rho})$, with $\mathbf{w} = (1/3, 2/3)$ and $\boldsymbol{\rho} = (2, 1)$.

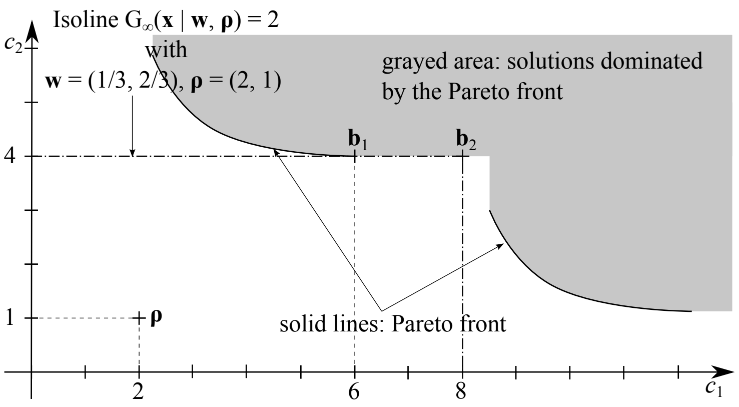


Figure 5.6. *Aggregation of the objectives with the Chebyshev distance from a reference point $\boldsymbol{\rho} = (2, 1)$: the minimum of $G_\infty(\boldsymbol{x}|\mathbf{w}, \boldsymbol{\rho})$ is 2 for $\mathbf{w} = (1/3, 2/3)$. The line segment $]\mathbf{b}_1, \mathbf{b}_2]$ represents the dominated vectors that are optimal for $G_\infty(\boldsymbol{x}|\mathbf{w}, \boldsymbol{\rho})$*

There are other aggregation methods such as the boundary intersection approaches [DAS 98, MES 03].

To obtain several solutions approaching the Pareto optimal set, the naive way consists of choosing different weight vectors as many times as desired and running a mono-objective optimization algorithm for each of them. The weight vectors have to be chosen to cover Pareto fronts as evenly as possible: this is not an easy problem. Authors have proposed aggregation methods able to generate a sample of non-dominated solutions close to the Pareto optimal set in a single run. MOEA/D [ZHA 07] is an example of an efficient aggregation-based algorithm to solve many objective problems.

The main difficulty with this kind of approach is obtaining a large enough diversity among the non-dominated solutions in order to ensure a good coverage of the Pareto front.

5.6.3. *Indicator-based approaches*

The indicators of quality discussed in section 5.3 can be used as an objective function to solve problems with many objectives. The measure of the hypervolume, or $\mathcal{S}$-metric, (section 5.3.1) is an excellent candidate because the maximization of this indicator ensures that the solutions obtained are on the Pareto front, and furthermore, that they are evenly distributed there. However, this metric requires evaluation in an exponential time in the number of objectives. Thus, SMS-EMOA [BEU 07] was proposed by Emerich *et al.* in 2005. Beume and Rudolph showed that the runtime of a generation is $\mathcal{O}(\mu^{c/2+1})$, where μ is the population size and c is the number of objectives. Therefore, the number of objectives cannot be too large, of the order of 8 maximum.

The algorithm HypE [BAD 11] estimates hypervolumes with Monte Carlo sampling. HypE is able to provide competitive results with test functions having up to 50 objectives.

Other indicators which are much easier to calculate than hypervolume can be used, but at the risk of performance deterioration because, unlike the $\mathcal{S}$-metric, they are not monotonic with Pareto dominance (section 5.3.1) [LI 15].

5.6.4. *Diversity-based approaches*

Diversity-based approaches propose diversity preservation mechanisms that do not impede the convergence towards the Pareto-optimal set. More details are given in [LI 14], which is one of the most recent references.

5.6.5. *Reference set approaches*

Aggregation methods presented above (section 5.6.2) use a reference point to compute a quality metric of solutions as a distance to minimize. Thus, reference points attract new solutions generated by the variation operators. Note that reference points could be defined without mention of aggregation methods, as long as they make it possible to define a quality metric. Using a set of well-chosen reference points can help to reduce the distance from the best individuals of the population to the Pareto-optimal set while increasing their diversity to improve their coverage of the Pareto optimal set.

For instance, NSGA-III [DEB 14] can generate in a normalized objective space $\mathcal{F}'$ a set of regularly spaced reference points on the hyperplane of equation $\sum_{i=1}^{c} f_i = 1$. Another alternative consists of taking into account a set of reference points supplied by the user to the optimization algorithm. Each reference point ρ defines a reference straight line passing through ρ and the origin of $\mathcal{F}'$. The closest solutions to these reference lines are good candidates for inclusion into the population for the next generation.

5.6.6. *Preference-based approaches*

Population sizes increase exponentially with respect to the number of objectives to ensure a good coverage of the Pareto front or the Pareto-optimal set. A way in which to limit the size of a population to an acceptable value is to focus exploration of the Pareto front in areas of interest that match the preferences of a decision maker. One method commonly used in multi-criteria decision-making is to weight the objectives according to the preferences of the decision maker: when an objective has more importance than another, its weight is greater. Other approaches are possible. Descriptions and references can be found in [LI 15].

5.7. Conclusion

Evolutionary algorithms are population algorithms able to provide a sample of non-dominated solutions in a single run when faced with multi-objective optimization problems. A decision maker can then choose the best alternative in this sample that fulfills its preferences.

The first effective approaches were based on Pareto dominance to reconcile the search for solutions close to the Pareto-optimal set as well as the even coverage of the Pareto front through a mechanism of diversity preservation. However, this approach cannot be used when the number of objectives is greater than 3 because it then requires populations of very large sizes, requiring excessive computation time, to have both dominated and non-dominated solutions in a population. When a problem has many objectives, other approaches, such as aggregating objectives or using quality indicators to define a fitness function, have yielded significant results for test problems with up to 50 objectives. However, increasing the number of objectives has the consequence of requiring a sample of solutions whose size is exponential according to this number of objectives, in order to cover the Pareto front without forgetting important parts. One way to avoid this drawback is to take into account the preferences of decision makers before or during the search for Pareto-optimal solutions, to explore only the interesting parts of the Pareto front from their points of view.

Multi-objective optimization faces many challenges. Active and productive research continues in this field.

6

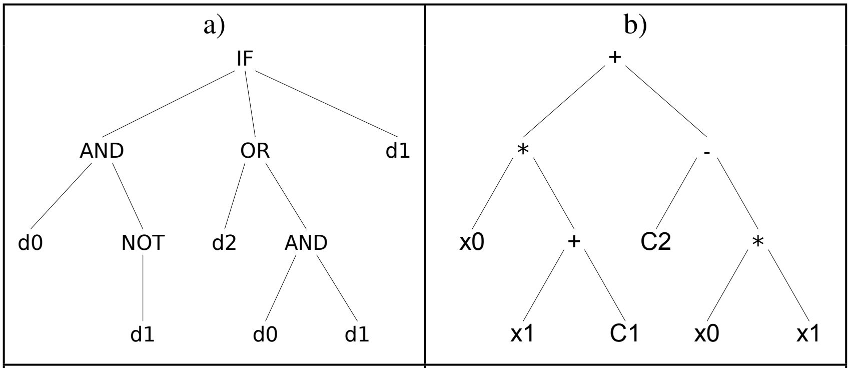
Genetic Programming for Machine Learning

6.1. Introduction

A wide variety of problems from many different fields can be formulated as a computer program induction that produces a desired output from a set of known input values. As examples of problems that can be considered as program induction, there are Automatic Programming, Planning, Sequence Induction, Symbolic Regression, Discovering Game-Playing strategies, Forecasting, Induction of Decision Trees, etc.

Approaching the program induction problems with evolutionary algorithms needs more powerful representations than that of standard GA with greater flexibility. Cramer introduced in 1985 the dynamic tree-based representation for Genetic Algorithms [CRA 85], in order to evolve sequential sub-programs written using a simple programming language. The evolution engine used was the Steady-State Genetic Algorithm (SSGA), whose task was not to find the optimal solution of an optimization problem but to discover the computer program that may solve the problem.

John Koza adopted the syntax tree representation in 1992 [KOZ 92] to define Genetic Programming (GP) as a new evolutionary algorithm to solve program induction problems. Its main objective was to evolve sub-programs in LISP language (Figure 6.1 (a)). He showed empirically that his approach makes it possible to discover the relevant programs for a large number of application examples, including the design of complex objects, such as electronic circuits, with an effectiveness significantly higher than chance would allow.



$\mathcal{N} = \{\mathrm{IF}, \mathrm{AND}, \mathrm{OR}\}$	$\mathcal{N} = \{\times, +, -\}$
$\mathcal{T} = \{d0, d1, d2\}$	$\mathcal{T} = \{x0, x1, \mathbb{R}\}$

Figure 6.1. *Example of tree solutions obtained by Genetic Programming where the search space is a set of LISP subprograms a), and where the search space is the space of functions representing polynomials with 2 variables b).*

Thanks to Koza's book, published in 1992 [KOZ 92], the application of genetic programming has expanded to the resolution of many types of problems, whose solutions can be represented by syntax tree structures, such as linear functions [NOR 94] (Figure 6.1(b)), graphs [TEL 95, RYA 98] and molecular structures [WAS 01]. Afterwards, several other approaches and representations were proposed for GP [POL 08].

This chapter presents the basics of the tree-based representation, which is the first GP implementation proposed by Cramer [CRA 85] and Koza [KOZ 92] (section 6.2). An introductory example in section 6.4 illustrates the preparatory steps needed for applying GP. This chapter also presents two other GP approaches using linear-based representation (section 6.5.1) and graph-based representation (section 6.5.2) and introduces some GP variants. An example of real-world application is given in the last section in order to demonstrate how effective GP can be in solving such a problem.

Why use Genetic Programming?

Genetic Programming is considered as the evolutionary technique having the widest range of application domains. It can be used to solve problems in at least three main fields: optimization, automatic programming and machine learning. In machine learning, many published works demonstrate that GP is applicable and effective to solve a surprising variety of problems. We summarize below a range of domains in which GP has proved useful.

Empirical Discovery and Forecasting

Empirical discovery involves finding patterns that relate a given input value to the associated output values for some observed systems in the real world. Once an accurate model has been established, it can be used to forecast future values in the system. This field includes some other sub-fields such as:

– Data Classification: Classification might be applied with supervised or unsupervised learning (known as clustering). With unsupervised learning, given a set of observations, the aim is to establish the existence of classes or clusters in the data. With supervised learning, classes are known and the aim is to establish a rule whereby it is possible to classify a new observation into one of the existing classes. Genetic Programming is widely used in both cases. It was successful in discovering classes/rules for a large variety of classification problems, and it outperforms well-known classification methods in several cases.

– Symbolic Regression: Symbolic regression builds a symbolic function that matches given input-output data as closely as possible. Symbolic regression is one of the most popular applications of genetic programming and an attractive alternative to standard regression approaches with real-life distribution, where other regression methods may fail [KOZ 10]. Its success is due to its flexibility in generating free-form mathematical models from observed data without any domain knowledge. A complete definition of symbolic regression is given in section 6.4.1 with an example of application.

Software agents

Jonh Koza, in his book [KOZ 92], demonstrated that GP could evolve an agent that takes multiple sub-goals into account. He experimented the program for the “Pac Man” controller, where the agent was a player with the goal of collecting the maximum number of points in a given period of time, with multiple sub-goals such as ghost avoidance. Thanks to this first successful demonstration, a wide variety of applications have been developed in this field such as:

– Evolution of Emergent Behavior: emergent behavior involves the repetitive application of seemingly simple rules that lead to complex overall behavior (e.g. robots);

– Multi-agent cooperation: multi-agent systems focus on systems in which many autonomous agents interact with each other. The agents can share a common goal and be cooperative, or their interactions can be selfish. GP was successfully applied to evolve cooperative multi-agent systems for many purposes (robotics, hard classification problems, etc.) [PAN 05].

6.2. Syntax tree representation

A Genetic Programming population is composed of a set of hierarchically structured computer programs represented as dynamic syntax trees. The sizes, the shapes and the contents of these computer programs can dynamically change during the evolution.

A syntax tree is composed of a set of leaves, called terminals ($\mathcal{T}$), and a set of nodes, called non-terminals ($\mathcal{N}$). The two sets $\mathcal{T}$ and $\mathcal{N}$ together form the primitive set of a GP system.

Using GP requires the definition of the two sets of nodes $\mathcal{N}$ and leaves $\mathcal{T}$ that define the search space. The components of these two collections depend on the problem. The non-terminal set may include arithmetic operations, mathematical functions, Boolean operations (e.g. AND, OR, NOT), conditional operators (e.g. If-Then-Else) and functions causing iteration (e.g. Do-Until). The terminals are, typically, variable atoms (i.e. representing the inputs) or constant atoms or functions with no explicit arguments. In this way, for real-valued functions, a solution is a syntax tree constructed from:

1) a set of non-terminal symbols, which can be some arithmetic operators ($\times, \div, -, +$) or some mathematical functions with arguments (sin, cos, ln,...);

2) a set of terminal symbols, which can be variables, universal constants or functions without arguments (rnd( ), time( ), etc.).

6.2.1. *Closure and sufficiency*

In order for GP to work effectively, the primitive set must respect two important properties: closure and sufficiency [KOZ 92]. The property of sufficiency requires that the sets of terminal and non-terminal symbols be able to represent any solution of the problem. This means that the set of all possible recursive compositions of the primitives must represent the search space. For example, the set $AND, OR, NOT, X_1, X_2, \cdots, X_N$ is a sufficient primitive set for Boolean function induction.

The property of closure implies that each node must accept, as an argument, any type and value that can be produced by a terminal or non-terminal symbol. This means that any leaf or sub-tree can be used as an argument for every node in the tree. In some cases, the closure can be easily satisfied as for Boolean function sets. In other cases, it is necessary to deal with some particular situations. For example, to achieve closure for arithmetic operators, we need to handle the case of the division by zero.

6.2.2. *Bloat control*

The shape of individuals with genetic programming is very different from those mentioned previously for other representations. The trees must, in particular, have a mechanism for regulating their sizes. Otherwise, they will have a tendency to grow indefinitely over the generations, consuming more memory and computing power unnecessarily. Eventually, an evolved program could even exceed the memory capacity of the host computer. This phenomenon is called *bloat*. The control mechanism can be simply implemented from additional parameters to the GP system, such as a maximum depth value for the trees or a maximum number of nodes. The genetic operators in the GP system must be modified to respect these constraints.

6.3. Evolving the syntax trees

With Genetic Programming, a syntax tree corresponds to a genotype for an Evolutionary Algorithm. To evolve a GP population, we need to define the variation operators and the parental and environmental selections as introduced in Chapter 1. All the selection approaches available for the standard EA are also available for GP since they are based essentially on the population fitness. However, initialization and variation operators are re-defined for the syntax

6.3.1. *Initializing the population*

With the tree-based representation, initializing the population does not follow the same rules as with the binary and real representations. Each tree is generated in two steps: first the nodes, then the leaves. The selection of the first component, which will be the label of the root node, is restricted to the function set $\mathcal{N}$. This restriction aims to avoid the generation of a degenerate tree structure consisting of a single terminal atom. Whenever the root is labeled, the construction of the hierarchical structure continues with a random selection (using a uniform random probability distribution) from both sets $\mathcal{N}$ and $\mathcal{T}$. If a terminal is chosen to be the label for any point, that point becomes a leaf of the tree. The shape of the final tree depends on the initialization approach. The simplest and earliest syntax tree initialization methods are:

– *Grow* method: the generated trees have irregular shapes; at each step, the selection is done in a uniform manner in the sets of nodes and terminals until the maximum depth is reached, below which only terminals may be chosen (Figure 6.2(a));

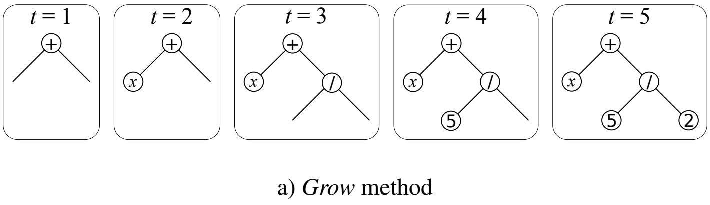
– *Full* method: the trees are balanced and full; for a given node, a terminal is chosen only when the maximum depth is reached (Figure 6.2(b));

– *Ramped Half and Half* method: given that the two previous methods do not offer a large variety of tree shapes and sizes, Koza [KOZ 92] proposed to combine them. With this method, half of the initial population is generated using *Full* and half is generated using *Grow*. The method uses a range of depth limits varying between 2 and the maximum depth. Currently, this technique is preferred to the two previous methods.

Algorithm 1 shows the pseudo-code of the procedure implementing the *Full* and *Grow* approaches in a recursive way.

6.3.2. *Crossover*

The traditional crossover with the syntax tree representation is the *sub-tree crossover*. It consists of an exchange of two sub-trees from the two individuals to cross, selected *a priori* among the more efficient. Therefore, they potentially contain interesting sub-trees. The crossover point in each parent tree is chosen randomly. An example of sub-tree crossover is illustrated in Figure 6.3.



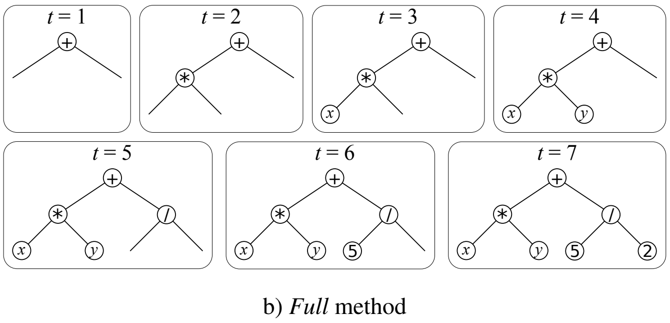


Figure 6.2. *Construction of a syntax tree with a maximum depth of 2, using the* Grow *method a) and the* Full *method b)*

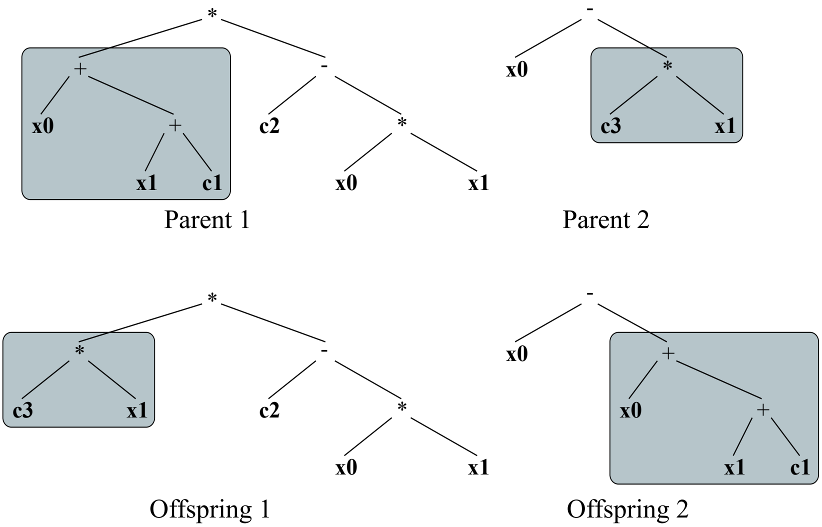


Figure 6.3. *Example of sub-tree crossover*

```
Parameters:
$\mathcal{N}$: function set
$\mathcal{T}$: terminal set
$d$: maximum allowed depth
$m$: initialization method (Full or Grow)
Result: expr: the generated expression in prefix notation

Procedure InitFullGrow($\mathcal{N}, \mathcal{T}, d, m$)
if $(d = 0)$ or $(m =$ Grow and $\mathcal{U}[0,1] < \frac{|\mathcal{T}|}{|\mathcal{T}|+|\mathcal{N}|})$ then
    // $\mathcal{U}[0,1]$: uniform random number in [0, 1]
    expr $\leftarrow$ choose a random element in $\mathcal{T}$
end
else
    func $\leftarrow$ choose a random element in $\mathcal{N}$
    for $i = 1$ to $arity$(func) do
        $arg_i \leftarrow$ InitFullGrow($\mathcal{N}$,$\mathcal{T}$,$d-1, m$)
    end
end
expr $\leftarrow$ (func, $arg_1, arg_2, ...$)
return expr
```

Algorithm 1: Recursive program of the *Full* and *Grow* methods

This general principle of crossover introduced by Cramer in 1985 [CRA 85] can be refined with different extensions to constrain the size of the generated offspring. Indeed, it is necessary to check the maximum depth for each syntax tree in the new population so that individual size does not become unnecessarily huge. If the cross-points chosen do not respect the limit size value, then recombination may not take place. The attitude adopted in this case is a parameter of the crossover. It will be at least one of the two following choices:

– select a new couple of parents and try to re-apply the crossover operator until two offspring respecting the size constraint are found;

– or even choose a new crossover point on the two selected parents until the resulting offspring satisfy the maximum depth constraint.

There are some other crossover operators proposed for tree-based GP, such as the *context-preserving crossover* and the *size-fair crossover* [LAN 00]. The *context-preserving crossover* attempts to preserve the context in which sub-trees appeared in the parent trees. It then restricts the crossover to take place only between sub-trees in similar locations (defined in terms of their node coordinates in their respective parent trees). Its objective is to avoid the spreadout of Building Blocks over the entire tree, as the regular crossover does. The objective of the *size-fair crossover* is to control the increase of the tree size along the evolution. It aims to solve the *bloat* problem without adding constraints or penalization to the algorithm. It starts in the normal way by selecting two parents and the crossover point in the first parent. The crossover point in the second parent is chosen such that the offspring size is in a computed interval. Otherwise, a terminal is always replaced by another terminal. However, even if this crossover is designed to reduce *bloat*, it increases GP complexity since it needs to compute all sub-tree sizes in each solution selected for crossing.

6.3.3. *Mutations*

The *traditional* GP system proposed by Koza [KOZ 92] does not use mutation operators. To ensure access to all primitives of the search language (e.g. LISP) and ensure genetic diversity, it uses a very large population size, to include a big quantity of the genetic material. GP mutation was introduced in 1996 by Angeline [ANG 96] in order to reduce the population size and thus the computation cost.

Due to the complexity of the GP syntax tree, multiple mutation operators have been proposed. Some of them are used for local search, but most of them could be applied for both local and global search. The most commonly used forms of mutation in GP are:

– *Sub-tree mutation*: the operator randomly selects a mutation point (node) in the tree and substitutes the corresponding sub-tree with a randomly generated sub-tree (Figure 6.4);

– *Point mutation* (also known as *cycle mutation*): a random node in the tree is replaced with a different random node drawn from the primitive set having the same arity (Figure 6.5);

– *Grow mutation*: it adds a randomly selected non-terminal primitive at a random position in the tree and adds terminals if it is necessary to respect the arity of the new node (Figure 6.6);

– *Shrink mutation*: a randomly chosen sub-tree is deleted and one of its terminals takes its place. This is a special case of sub-tree mutation that is motivated by the desire to reduce program size (Figure 6.7).

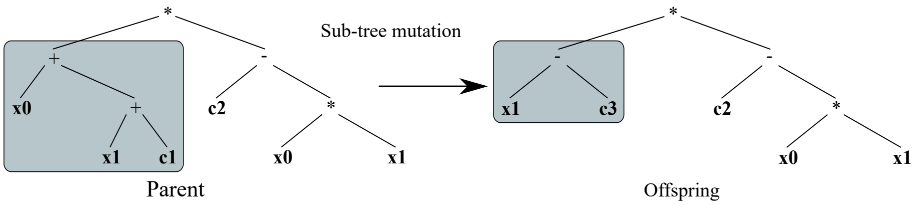


Figure 6.4. *Example of sub-tree mutation*

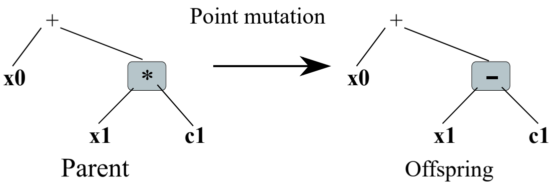


Figure 6.5. *Example of Point Mutation*

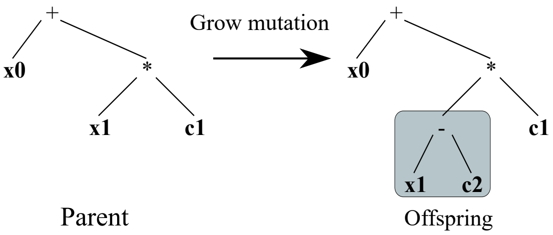


Figure 6.6. *Example of Grow mutation*

In the case where the leaves of the tree can take numeric values (constants), other mutation operators are introduced, such as:

– the Gaussian mutation: it mutates constants by adding a Gaussian random noise [ANG 96];

– optimized constant mutation: it tunes the solution by trying to find the best values for constants within the tree. It uses a numerical optimization method to reach the nearest local optimum, such as a *hill climber* [SCH 96b] or partial gradient ascent [SCH 95].

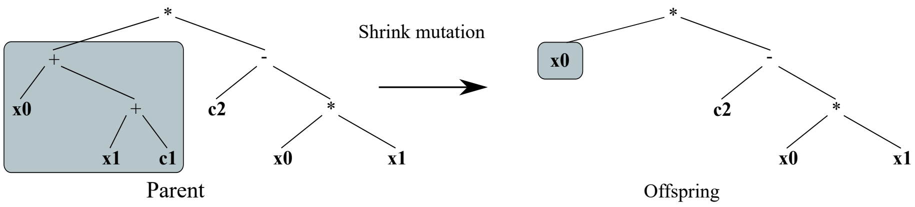


Figure 6.7. *Example of Shrink mutation*

6.3.4. *Advanced tree-based GP*

Because of the closure and sufficiency assumptions, the standard GP imposes restrictions in the search space. For example, it is not possible to handle a mixture of data types. Thus, some advanced concepts were introduced to extend the GP search process for a complex problem. For this purpose, Koza introduced the Automatically Defined Functions (ADFs) to evolve reusable components [KOZ 94]. Moreover, Montana introduced the Strongly Typed GP [MON 95] to handle multiple data types within a single tree.

6.3.4.1. *Automatically defined functions*

When evolving GP tree-based populations, Koza observed that some sub-trees appear repeatedly within the best individuals. He then introduced the Automatically Defined Functions (ADFs) [KOZ 94] to evolve reusable components in GP genotypes (Figure 6.8). Useful sub-trees, such as subroutines, functions and classes, are then identified, encapsulated into modules, and reused as single units in the evolutionary process.

To evaluate a solution, the main program calls some ADF programs, which might also call other ADFs programs. Otherwise, the same ADF program might be called more than once by the main program.

6.3.4.2. *Strongly Typed GP*

With Strongly Typed GP (STGP), the representation of the components in the terminal and non-terminal sets is extended. A type (float, vector, matrix, etc.) is assigned to each variable or constant in the terminal set. Similarly, a type for each argument and for the value it returns is assigned to each operator or function in the non-terminal set. This representation enables the handling of multiple data types in the same parse tree. However, to evolve legal Parse tree, additional constraints must be handled by the STGP operators:

– the root node of the tree returns a value of the type required by the problem;

– each non-root node returns a value of the type required by the parent node as an argument.

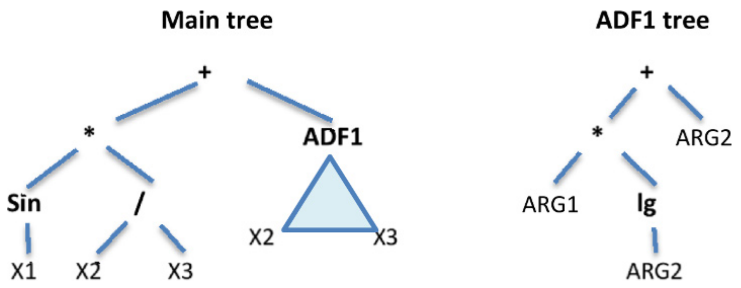


Figure 6.8. *ADF example. To evaluate the main tree, the ADF1 tree is called with X2 and X3 as values for the arguments ARG1 and ARG2, respectively*

6.4. GP in action: an introductory example

There are five major steps in preparing to use genetic programming, namely determining:

1) the set of terminals;

2) the set of primitive functions;

3) the fitness function;

4) the parameters for controlling the run;

5) the method for producing a result and the criterion for terminating a run.

This section details the application of the aforementioned five preparatory steps with an introductory example. A simple genetic programming algorithm is designed to solve a symbolic regression problem. We first define the symbolic regression and the example to study. We then present the impact of the choice of the primitive set and the controlling parameter on the performance of the algorithm.

6.4.1. *The symbolic regression*

Regression analysis have been studied substantially in the fields of statistics and signal analysis. However, John Koza [KOZ 92] showed that Genetic Programming can be used advantageously to solve symbolic regression problems. Each tree in the population may represent a mathematical expression.

Given a database containing a set of N pairs of vectors $(\mathbf{x}_j, \mathbf{y}_j)$ for $j \in [1, N]$, *the symbolic regression* consists of discovering a symbolic expression $\mathcal{S}$ able to map the input vector $\mathbf{X} = (x_1, .., x_N)$ to the real target value $\mathbf{Y} = (y_1, ..., y_N)$. *A priori*, there is no constraint on the structure of the search expression $\mathcal{S}$. For a vector $\mathbf{x}_j$, the expression $\mathcal{S}$ enables the computation of $\hat{\mathbf{y}}_j = \mathcal{S}(\mathbf{x}_j)$ whose gap with $\mathbf{y}_j$ must be minimized for any j by modifying the structure of $\mathcal{S}$.

Forecasting performance: fitness

For Genetic Regression, an individual in the population is a syntax tree. To construct the symbolic expression, GP randomly selects and combines independent variables from the terminal set and operators from the primitive function set. Each syntax tree is evaluated according to a user-defined fitness function that measures the mapping error between the input and output data.

This error is called Forecasting Performance and is used by GP as the fitness function for the evolution process. The Mean-Squared Error (MSE) is widely used. It computes the mean-squared error between the estimated value ($\hat{y}_i$) and the real target value (y_i):

$$MSE = \frac{1}{N} \sum_{N}^{1} (y_i - \hat{y}_i)^2 \qquad [6.1]$$

where N is the number of entries in the training data sample.

There are other popular metrics based on the error between the target and estimated values, such as the *Root Mean-Squared Error*: *RMSE* $= \sqrt{\frac{1}{N}\sum_N^1 (y_i - \hat{y}_i)^2}$ and the *Mean Absolute Error*: *MAE* $= \frac{1}{N}\sum_N^1 |y_i - \hat{y}_i|$. From a statistical point of view, these metrics do not differ from the MSE.

Many other metrics are used for symbolic regression. They are less popular but might have an effect on the search. For example:

– the Mean Relative Error (MRE) is used to have a cumulative measure of the forecasting error:

$$MRE = \frac{1}{N}\sum_N^1 \frac{|y_i - \hat{y}_i|}{y_i}$$

– the Ratio of Squared Errors (RSE) shows an improvement over random walk models:

$$RSE = \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=2}^N (y_i - y_{i-1})^2}$$

6.4.2. *First example*

For this first example, we try to find the function $\sqrt{a^2 + b^2}$. The input vector contains only 10 components where the values of a_j and b_j are selected randomly in the interval $[1, 30]$. The input data $X = (a_j, b_j)$ and the target data Y are given in the following table.

Input X	(1, 7)	(4, 24)	(5, 15)	(8, 19)	(11, 15)	(14, 23)	(16, 12)	(17, 5)	(18, 25)	(29, 20)
Target Y	7.07	24.33	15.81	20.61	18.6	26.92	20	17.72	30.8	35.22

Table 6.1. *Input and target datasets for the introductory example*

The first two steps to prepare a GP system are the definition of the terminal and non-terminal sets. Usually, the terminal set includes input variables. In the case of our example, it is composed of the two variables a and b. The non-terminal set (or function set) includes basic mathematical operators such as addition, subtraction, multiplication, protected division and protected square root.

The third preparatory step consists of defining the fitness measure. For this simple example, we use the most popular fitness measure for symbolic regression that is the MSE function (equation 6.4.1).

Terminal set (leaves)	a, b
Non-terminal set (nodes)	$+, -, *, \%, Sqrt$

Table 6.2. *Terminal and function sets for the introductory example.*

The fourth step concerns the parameter setting. For the first experiments, we set the GP parameters according to Koza's suggestions ([KOZ 92]), except for the population size and the mutation rate. Population size is a major control parameter in GP. According to Koza, the greater the number of individuals in the population (hundreds, thousands or more), the higher the probability of finding the global optimum. In our case, since the desired function is not complex, we do not need a large population. The population size is set to 300, which has proven to be sufficient for our problem. We use the Steady-State replacement (section 1.3.6.5). We also use four mutation operators applied with very low probabilities. Note that Koza suggests not to use mutation with tree-based GP. The complete parameter setting is summarized in Table 6.3. Other experiments are then performed by varying the values of some parameters such as mutation and crossover probabilities.

For the final preparatory step, we need to specify a termination condition. For this example, we will make do with a simple condition by setting the maximum number of generations to 10 as given in Table 6.3.

6.4.2.1. *Experimental behavior*

With the conditions set up, GP has no difficulty in finding the optimal solution in one generation. If the evolution continues for further generations, the whole population converges to the optimal solution. However, GP might lose its efficiency when some ingredients or some experimental settings vary. The following experiments aim to demonstrate the importance of the preparatory stages for GP application. For each case presented in the following sub-sections, at least 11 runs are done before drawing conclusions.

Population size (P)	300
Maximum number of generations (G)	10
Maximum depth of new individual	4
Maximum depth of the tree	8
Tournament size	4
Crossover probability (p_c)	0.7
Mutation probability(p_m):	0.1
Branch mutation	0.025
Point mutation	0.025
Grow mutation	0.025
Shrink mutation	0.025

Table 6.3. *GP parameter settings for the introductory example*

6.4.2.1.1. *The function set*

The choice of components of the program determines the search space in which genetic programming searches. The function set for our example is based on the fact that output will be a simple algebraic function of (a, b). If this choice is wrong, then it may not be possible to produce a correct program. Otherwise, it is possible to extend the function set to be sure to include all the possibilities. However, this can considerably increase the search space size, which may slow down the system.

We tried to carry out further tests under the same settings summarized in Table 6.3; however, we have expanded the function set with two new features: exponential function and logarithmic function. GP is no longer able to quickly find the optimal solution from the first generation. Several trials are done, and the program was not able to find the optimal solution before 400 generations on average, which can be considered costly for such a simple problem. If we add two other trigonometric functions such that the function set becomes: $\mathcal{N} = \{+, -, *, \%, \mathrm{sqrt}, \log, \exp, \cos, \sin\}$, at least 850 generations are needed to find the optimal solution. However, the slowdown of the GP caused by the extension of the search space could be overcome with a larger population size. Indeed, an increase of this parameter to 600 allows GP to recover its performance and find the optimum in less than 30 generations. Thus, it is clear that with an enlarged function set, the exploration ability should be enhanced. Increasing the population size could be sufficient. Otherwise, it could be useful to increase the mutation probability to increase diversity within the offspring population.

6.4.2.1.2. *Design of the fitness function*

The choice of the fitness function for symbolic regression problems is critical because this value is the basis for the selection strategy. If some individuals have a very high fitness in relation to the others, more fit individuals would quickly dominate and result in premature convergence. Thus, the fitness function must be carefully designed such that it reflects the real performance of the individuals.

The statistical functions in the same category of the Mean-Squared Error can be used as fitness measures for symbolic regression problems. For example, for this case, when MSE is replaced with MAE or RMSE with the same experimental settings, GP is still able to find the optimum from the first generation. Otherwise, if the orders of magnitude in fitness measures of solutions are very different, it is recommended to use a normalized measure.

6.4.2.1.3. *Parameter settings*

With GP, different sets of parameter values can cause significant difference over the effort required to obtain an optimal solution. Experiments in this section deal with the two parameters that usually impact the diversity of the population: the mutation and crossover probabilities.

Decreasing the crossover probability may dramatically decrease the performance of the algorithm. It may either increase the number of generations needed to converge or bring the population to a local optimum. Let us test the GP efficiency with different values of p_c varying from 0 to 1. The mutation probability is still unchanged ($p_m = 0.1$). We noted the two following observations:

– when $p_c > 0.7$, the GP is usually able to quickly find the optimal solution in few generations (less than 30 generations);

– when $p_c < 0.3$, the GP converges to a local optimum for practically all runs. This behavior is not surprising. The lack of diversity associated with the elitism applied in the replacement step inevitably leads to a premature convergence.

As explained by Koza in his book [KOZ 92], the main operator for genetic programming is the crossover operator. According to his recommendations, with a large population size and adequate crossover rate, GP does not need mutation. Let us try to run the GP system without crossover and with a mutation rate varying from 0 to 1. The first observation is that GP converges quickly to a local optimum with $p_m \leq 0.1$. This is caused by the lack of diversity in the population since there is no solution crossing. Otherwise, GP is not able to converge with a high mutation rate (i.e. $p_m = 0.8 \cdots 1$). Indeed, best solutions are not well exploited and could easily be lost after the mutation step. However, with a mutation rate varying from 0.2 to 0.7, the optimum is found in few generations. Thus, the implemented mutation methods applied with an adequate probability are able to ensure the exploration of the search space in parallel to the exploitation of the best solutions. Nevertheless, this performance can not be expected when using fixed-length sub-tree mutation such as *point mutation*. Only variable-sized mutation may prove competitive with a crossover operator for exploring the search space, such as the *sub-tree mutation* or *grow mutation* (section 6.3.3).

Tuning parameters for an evolutionary algorithm is generally a difficult task. For tree-based GP, besides Koza's recommendations [KOZ 92], several other comparative studies are published [LUK 97, LUK 98, WHI 09]. According to these studies and our own experience in this domain, we suggest using a population size between 400 and 500, to keep a reasonable computational cost [POL 08], but it should be adjusted according to the problem to solve. It is also suggested to combine crossover and mutation with a crossover probability varying from 0.5 to 0.7 and a lower mutation probability. For mutation, it is recommended to use fixed-sized and variable-sized mutations to ensure both exploitation and exploration of the search space.

6.5. Alternative Genetic Programming Representations

The most common form of GP employs trees for its representation. However, multiple other representations, which make differences between the genotype and phenotype of an individual, have been introduced. Thus, the phenotype can be encoded using either linear sequences or graphs. For each representation, several variants were proposed. We tried to summarize the different GP implementations in Table 6.4 based on one of the three representations: tree-based representation, linear-based representation and graph-based representation. In this section we present three of these implementations that have proven successful in practice: LGP (Linear GP), GE (Grammatical Evolution) for linear-based representation and CGP (Cartesian GP) for graph-based representation. The tree-based representation is presented in section 6.2. For further readings about the other GP implementations, we suggest [POL 08, BRA 07] and [LAN 12].

Tree-based representation	Standard GP [KOZ 92]
	Strongly Typed GP (STGP)[KOZ 94]
	Automatically Defined Functions (ADF)[KOZ 92]
Linear-based representation	Linear GP (LGP) [BAN 93, PER 94]
	Grammatical Evolution (GE) [O'NE 01, O'NE 03]
	Gene-Expression Programming (GEP)[FER 06]
	Multi-Expression Programming (MEP) [OLT 03]
Graph-based representation	Cartesian GP (CGP) [MIL 11]
	Parallel Distributed GP (PDGP) [POL 97]
	Parallel Algorithm Discovery and Orchestration (PADO)[TEL 96]
	Multiple Interacting Programs (MIPs)

Table 6.4. *Most used GP variants for the tree-based, linear-based and graph-based representations*

6.5.1. *Linear-based GP Representation*

Linear-based GPs (LGPs) are variants of the standard GP evolving linear genotypes (Figure 6.9). The first implementations of linear-based GP were motivated by the idea of representing the GP solution as a linear computer program that can be executed in a sequential way [BAN 93, PER 94]. Linear GP were also introduced to speed up the GP system by avoiding the interpretation of the tree-shaped programs [NOR 94, NOR 99].

Banzhaf is the first author who proposed a linear representation to encode the GP genotypes within the Linear GP system (LGP) [BAN 93]. Afterwards, several other linear variants of GP have been proposed, such as the Grammatical Evolution (GE) proposed by Ryan, Collins and O'Neill [O'NE 01, O'NE 03], the Gene Expression Programming (GEP) proposed by Ferreira [FER 06] and the Multi-Expression Programming (MEP) proposed by Oltean and Dumitrescu [OLT 02, OLT 03]. Like LGP, GEP and MEP, use integer and real numbers for individual encoding. However, GE uses binary strings. Otherwise, MEP has an additional ability that is to encode several solutions with the same genotype. A common advantage of the linear-based GP is that it avoids bloat. Indeed, MEP and GEP use fixed-length genotypes. LGP uses variable-size genotypes that are limited to a maximum number of instructions (genes). In the following sub-sections, we present the basics of the evolution scheme of LGP and GE.

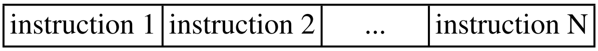


Figure 6.9. *Typical linear GP representation*

6.5.1.1. *Linear GP (LGP)*

The basic LGP program uses linear sequences of instructions (e.g. instructions of an imperative programming language like *C*). Instructions operate on indexed variables (also called registers) or constants from predefined sets. An instruction may operate on one or two register(s), and the result is put in another register. Let $r[i]$ be a register of index i. The different forms of an instruction are:

1) $r[i] = r[j]\ op\ r[k] \Rightarrow$ instruction operating on two registers, where *op* is a binary function from $\mathcal{N}$;

2) $r[i] = op(r[j]) \Rightarrow$ instruction operating on one register, where *op* is a unary function from $\mathcal{N}$;

3) $r[i] = r[j]\ op\ c \Rightarrow$ instruction operating on one register and one constant.

A LGP phenotype is a variable-length sequence of instructions respecting one of the three forms given above. However, the genotype is a vector of integers grouped on sequences of three or four integers representing the function label and the output/input register indexes. For example, the first and second phenotype forms are encoded as follows:

$$r[i] = r[j] opr[k] \Rightarrow (\textit{op-label}, i, j, k)$$
$$r[i] = op(r[j]) \Rightarrow (\textit{op-label}, i, j,)$$

where *op-label* is the label of the function *op* in the function set $\mathcal{N}$. Then, the instruction $r[4] = r[2] + r[5]$ is encoded with the vector $(1, 4, 2, 5)$, where the operator $+$ has 1 as label.

6.5.1.1.1. Evolving LGP genotypes

A typical crossover operator for LGP is the swap operator that swaps two code fragments between two parents. It acts in a similar way to the one-point crossover for the binary representation used by Genetic Algorithms. It is applied independently of the machine code in the parents to cross. The swapped fragments are defined thanks to two randomly chosen crossover points. Note that the crossover can change the program's lengths since the blocks are typically of different lengths. Two variants are possible: the Typical crossover that generates a single offspring (Figure 6.10) and the Homologous crossover that generates two offspring (Figure 6.11).

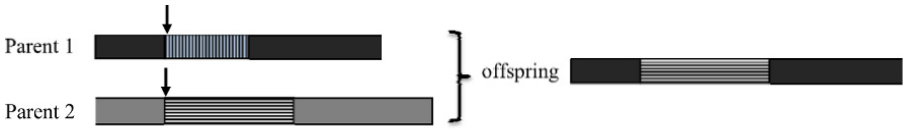


Figure 6.10. *Typical LGP Crossover*

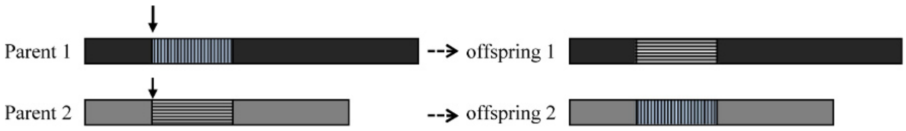


Figure 6.11. *Homologous LGP Crossover*

Several mutation operators have been defined for linear representation that respect the main aspects of the well-known mutation operators for evolutionary algorithms and they have been adapted to the linear representation such as the point mutation or the swap mutation. These operators act on machine code instructions, but should respect the validity of the phenotype (all instructions belong to the function set, the register and constant values are within the predefined intervals).

With point mutation (or micro-mutation) [LAN 05], an instruction is randomly selected, and then, an X-OR operation performed with a second randomly generated instruction. With the swap mutation, an arbitrary pairwise swap is performed between two instructions in the same individual.

Further operators were proposed for LGP. A detailed description with examples of LGP applications could be found in [BRA 07].

6.5.1.1.2. LGP strengths and weakness

The main advantage of LGP compared to the tree-based GP is the gain of speed. Since LGP evolves low-level programs, these ones are run directly by the computer processor, which avoids a program interpreter for computing the fitness function. However, with LGP, it is difficult to define the appropriate number of registers needed for solving a given problem.

6.5.1.2. *Grammatical Evolution (GE)*

The Grammatical Evolution (GE) [O'NE 01, O'NE 03] is a grammar-based Genetic Programming using a different paradigm introducing a genotype-phenotype mapping. The genotype is variable-length, linear binary vectors of integers (codons) encoded with binary blocks. Before evaluating an individual, the corresponding genotype is mapped to a phenotype representing a program in the language specified by the given grammar. The phenotypes are temporary, and they are created only for the evaluation step.

6.5.1.2.1. The context-free Grammar

To describe programs (individuals), GE needs a context-free Grammar in BNF (Backus-Naur form). It is defined by the 4-tuple ($\mathcal{T}$, $\mathcal{N}$, R, S), where:

– $\mathcal{T}$: Terminal symbols that are the lexicon of the language;

– $\mathcal{N}$: Non-terminal symbols that can take different values;

– R: A set of production rules that defines what symbols are replaced in the set of non-terminals. Every production rule defined by $n \rightarrow v$ is realized by replacing the non-terminal symbol $n \in N$ with the symbol $v \in V$, where V is a sequence of terminal and/or non-terminal symbols ($V \subseteq (\mathcal{T} \cup \mathcal{N})$;

– S: A start symbol that determines a non-terminal from which the generation of the expression starts.

Following an example of a grammar and these collections adopted from [O'NE 01]:

$$\mathcal{N} = \text{expr, op, pre-op}$$

$$\mathcal{T} = \sin, +, -, /, *, X, 1.0, (,)$$

$$S = \text{expr}$$

and R can be represented as:

(0) <expr>	=	<expr> <op> <expr>	(0)
	|	(<expr> <op> <expr>)	(1)
	|	<pre-op>(<expr>)	(2)
	|	<var>	(3)
(1) <op>	=	+	(0)
	|	$-$	(1)
	|	/	(2)
	|	$*$	(3)
(2) <pre-op>	=	sin	(0)
(3) <var>	=	X	(0)
	|	1.0	(1)

To understand the GE mapping procedure, let us try to decode the genotype (00000110 00001001 00000111 00001100) using the grammar given above. First, the genotype is translated into an integer form. We then obtain (6, 9, 7, 12). The start symbol is S = <expr>. We have four possibilities for the entry <expr>. We read the first element e in the genotype (i.e. $e = 6$) and then the rank of the production rule to choose is equal to e *modulo* the number of possibilities: $6 \bmod 4 = 2$. The symbol S becomes S = <pre-op>(<expr>). To decode <pre-op>, the same mechanism is used with the second value in the genotype, which needs first to compute $9 \bmod 1 = 0$, and the expression S becomes $S =$ sin<expr>. Next, to decode <expr> again, the entry in the rank given by $7 \bmod 4 = 3$ is chosen. The expression becomes $S =$ sin<var>. The last step determines the value of <var> by computing $12 \bmod 2 = 0$. The first possibility is chosen for the entry <var> and then we obtain the final expression: $S = \sin(X)$. Note that the mapping procedure of GE uses a recursive approach to treat the codons sequentially.

6.5.1.2.2. Evolving GE genotypes

The GE phenotype are binary strings to which standard GA operators, such as one-point crossovers, could be applied. However, the GE genotypes have a variable length. Therefore, to cross two genotypes, the cross-point should be chosen on the boundaries of the codons.

The mutation operator employed by GE is also similar to the one used for the binary encoding. This operator randomly flips one or several bits in a genotype and is applied according to a small mutation probability.

Two other operators, *duplicate* and *prune*, have been proposed for the GE system [RYA 98]. By duplication, a randomly chosen sequence of genes is copied into the position of the last gene of the genotype. The number of genes to be duplicated is picked randomly. Otherwise, the GE phenotype would not necessarily use all genes in the genotype. For example, to generate the expression $y - x * x$, only five genes are needed. The remaining genes are *introns*. The *prune* operator is usually applied for reducing the number of introns.

6.5.1.2.3. GE strengths and weaknesses

GE provides a general and natural way to evolve programs thanks to the context-free grammar. Compared to the tree-based GP, GE has the advantage of being able to change the mathematical expression representation (prefix, infix, postfix, etc.) by simply changing the grammar. However, the weakness of GE lies in the mapping procedure that can provide invalid solutions.

6.5.2. *Graph-based Genetic Programming Representation*

Since trees are special types of graphs, several extensions of GP have been proposed that evolve programs like graphs, where genotypes are represented by directed acyclic graphs. The Parallel Algorithm Discovery and Orchestration (PADO) proposed by Teller in 1996 [TEL 96] was the first implementation of a graph-based GP. PADO evolves programs represented by graphs in a specific structured language.

Poli [POL 99] proposed Parallel Distributed Genetic Programming (PDGP) which is a graph-like representation evolved for highly parallel programs, with nodes associated with functions and terminals and edges associated with the flow of control and results. PDGP evolves programs using the variations operators extended with a syntactic correctness. In its simplest form, PDGP is a generalization of standard GP where edges are directed and unlabeled. When more complex representations are used, PDGP can evolve neural networks, finite state automata, recursive transition nets, etc. Poli *et al.* [POL 97] have shown that this representation enables a fine control over the degree of parallelism and the reuse of partial results by the evolving programs.

Angeline proposed Multiple Interacting Programs (MIPs) [ANG 97] based on a representation that is a generalization of a recurrent neural network modeling any type of dynamic system. MIPs evolve complex behaviors using a set of independent "programs" that interact to realize a global behavior. Each program in a given set is unique and stored in the form of a parse tree.

Miller proposed Cartesian Genetic Programming (CGP) [MIL 99] where, like PDGP, genotypes represent graph-like programs. It is called "Cartesian" because it uses a two-dimensional grid of computational nodes implementing directed acyclic graphs. In this graph, nodes representing functions have many inputs and one output. The following sub-section presents more details about CGP.

6.5.2.1. *Cartesian GP (CGP)*

CGP was initially proposed by Julian Miller in 1999 [MIL 99, MIL 00] where a program is represented as a directed acyclic graph, the nodes of which are placed on a 2-D grid. A genotype is a list of integers that represent the program primitives and how they are connected together. Each integer encodes some information such as from where a node gets its data, what operations the node performs on the data and where the output data is required by the user.

Cartesian GP cannot suffer from genotype growth (*bloat*), as the genotype is of fixed size. However, the size of the phenotype (in terms of the number of computational nodes) can be anything from zero nodes to the number of nodes defined in the genotype. The genotype-phenotype mapping used in CGP is one of its defining characteristics.

A CGP node encoding a function has n inputs, which is the maximum number of arguments of the function, and an output. Inputs are indexes of the nodes providing arguments for the function of the current node. Similarly, the output is the index of the current node that may be used as input by other nodes. A node with its inputs and output is depicted in Figure 6.12.

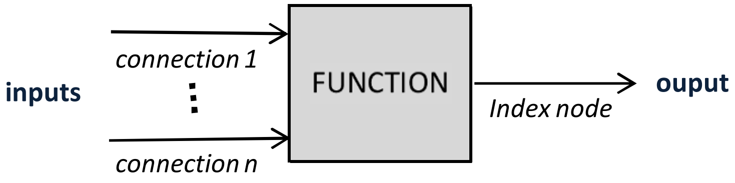


Figure 6.12. *Typical form of a CGP node*

If the function has m arguments with $m < n$, then only the first m connections are taken into account. Obviously, the function cannot have more than n arguments. Let us take a CGP case with two terminals $\mathcal{T} = \{x_1, x_2\}$ and four functions $\mathcal{N} = \{+, -, *, \cos\}$, presented as a 2×3 architecture. Figure 6.13 represents an example of CGP solution in the posed architecture. The program encoded is:

$$*((+x_1, *(x_1, x_1)), (+(x_1, x_2)))$$

CGP has an important parameter that defines the number of columns in the 2-D grid preceding the current node where it can get its inputs. This parameter, called *levels-back*, controls the graph connectivity, and then its topology [MIL 11]. Setting *levels-back* to 1 produces highly layered graphs, in which calculations are carried out column by column. For the example of Figure 6.13, this parameter is set to 2. If it was equal to 1, then input 3 of node 8 (final node) becomes invalid, since it can only have node indexes from the precedent column (i.e. nodes 5 or 6) as inputs.

The graph representation is only used for phenotype. CGP uses arrays of integer values to encode a genotype. The functions in the non-terminal set $\mathcal{N}$

are then labeled with integer values. Each node is encoded with its connection indexes and its function label as follows:

$$(input_1, ..., input_n, function\_label)$$

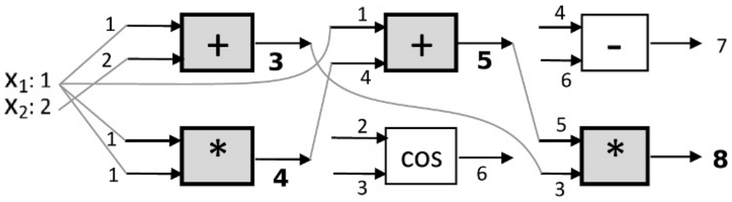


Figure 6.13. *An example of the CGP program with a 2×3 architecture. The program has two inputs (1,2), four functions (+,-,*,cos) and one output (8). The bold squares represent connected nodes*

To define the genotype of the above example, let us give labels to the function set: $\{(+) \rightarrow 1, (-) \rightarrow 2, (*) \rightarrow 3, cos \rightarrow 4\}$. The CGP program is then encoded as presented in Figure 6.14. The last value is the index of the output.

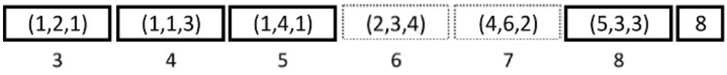


Figure 6.14. *Example of CGP genotype. The nodes used by the program are drawn with solid lines*

6.5.2.1.1. Evolving CGP genotypes

CGP uses basic mutation methods to evolve its genotypes. The main operator implemented for CGP is the *point mutation* that randomly chooses a gene, i.e. an integer of the genotype, and changes it to another valid random value. Two cases are possible according to the nature of the chosen gene g_m:

– if g_m is a function label, then it is replaced by a random element from the non-terminal set $\mathcal{N}$;

– if g_m is a node index associated to an input, then a random value is chosen from the output of any previous node in the genotype or from the terminal set $\mathcal{T}$.

Traditionally, CGP has always only used mutations to evolve populations. However, Clegg *et al.* proposed a crossover to speed up the convergence [CLE 07]. The results of their approach appear promising, but further work is required on a range of problems in order to assess its capabilities.

6.5.2.1.2. CGP strengths and weaknesses

CGP offers several advantages over other GP approaches. Unlike trees, there can be more than one path between any pair of nodes. This enables the reuse of intermediate results. It can also have multiple outputs, and thus, it is able to solve many types of problems. CGP is easy to implement, and it is highly competitive compared to other GP methods [WIL 08]. However, the main difficulty when applying CGP lies in setting its three main parameters: the number of columns, the number of rows of the 2-D grid and the *levels-back*. Below, we present an application of CGP to a simulated real-word classification problem.

6.6. Example of application: intrusion detection in a computer system

Section 6.4 has shown how easy it is to apply Genetic Programming to Symbolic Regression problems. We have chosen a classification problem as the second category of GP application. Several research papers explore the feasibility of applying GP to multi-category pattern classification problems. Here, we propose a CGP-based approach to design classifiers for an Intrusion Detection problem.

The major problem faced by an Intrusion Detection System (IDS) is the large number of false-positive alerts, i.e. normal behaviors mistakenly classified as alerts. There are different attack types that can be grouped into one of these four categories: User to Root (U2L); Remote to Local (R2L); Denial of Service (DoS) and Probe [TAV 09]. Many efforts in the Machine Learning field are made to improve IDS performance. The goal is to implement an IDS able to correctly classify network connections and to detect novel attacks with unknown signatures. Several machine-learning techniques have been applied to this classification problem (Neural Networks, Decision Trees, the K-nearest neighbor algorithm, Gaussian classifiers, fuzzy systems, etc.). Often, the patterns obtained offered an acceptable level of misuse detection performance for only two attack categories, namely Probing and DoS. The aim of using Genetic Programming for this problem is to produce less complex computer programs with a higher global performance compared to other machine-learning algorithms [SAB 03] and data-mining methods [ELK 00, CHA 13].

6.6.1. *Learning data*

The learning data used in this application was first created for the competition held at the Fifth International Conference on Knowledge Discovery and Data Mining (KDD-99), and is about intrusion detection problems. The original training dataset contains 5 million records about connections from simulated traffic and their labels. Another set was called the corrected set, constructed in the same way, and is used as the test dataset. The training process is run on the derived “10% KDD-99” dataset to learn how to classify normal connections and attacks. The best individual of each run is then tested on the test dataset. Both datasets are presented in Table 6.5.

	Normal	DoS	Probe	R2L	U2R	Total Attacks	Total Patterns
10% Training Set	97278	391458	4107	1126	52	396743	494021
Test Set	60593	229853	4166	16347	70	250436	311029

Table 6.5. *KDD-99 dataset composition*

Sampling training data

The specific interest of this work lies in identifying a solution to the problem of efficiently training with a large data set using CGP. To this end, we propose the use of sampling techniques that aim to reduce the original size of the training data set by substituting it with a much smaller representative subset. The main objective is that the selected subset gives the CGP the opportunity to produce an effective classifier. The test set is used to estimate how well the classifier will generalize the unseen data. For this example, two well-known dynamic sampling methods are selected: Random Subset Selection (RSS) [GAT 94] and Dynamic Subset Selection (DSS) [GAT 94]. A complete comparative study of the different sub-sampling techniques available in the literature applied on the intrusion detection problem can be found in [HMI 16b, HMI 16a]. The definitions below come from [HMI 16b]:

– RSS: the selection of fitness cases (data entries) is based on a uniform probability among the training subset;

– DSS: this technique is intended to preserve training set consistency while alleviating its size by keeping only the difficult cases with ones not selected for several generations. Each dataset record is assigned a difficulty degree $D_i(g)$ and an age $A_i(g)$ starting with 0 in the first generation and updated in every generation g. The difficulty is incremented per one individual misclassification and reset to 0 if the fitness case is selected. The age is equal to the number of generations since last selection. Then, in each generation, it is incremented when the fitness case has not been selected and reset to 0 otherwise. The resulting weight W of the i^{th} case is calculated with this sum:

$$\forall i \in \{1, ..., T\}, \quad W_i(g) = D_i(g)^d + A_i(g)^a. \tag{6.2}$$

where T is the size of the full dataset, d is the difficulty exponent and a is the age exponent. The selection probability $P_i(g)$ is biased by fitness case difficulty and age:

$$\forall i \in \{1, ..., T\}, \quad P_i(g) = \frac{W_i(g) * S}{\sum_{j=1}^{T} W_j(g)} \tag{6.3}$$

where S is the target subset size.

6.6.2. *GP design*

Usually the tree-based GP is efficient to solve symbolic regression problems, as described in section 6.4. For this application, we used a graph-based representation implemented with the Cartesian GP. Otherwise, as introduced in section 6.4, applying GP requires five preparatory steps. Below we summarize the design of the CGP program according to these five steps.

Performance metrics

By the end of each run, the best individual based on the fitness function is tested on the whole dataset and then the test dataset. Results are recorded in a confusion matrix, from which accuracy, True Positive Rate (TPR) and False Positive Rate (FPR) are calculated:

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ patterns}. \tag{6.4}$$

$$TPR = \frac{True\ Positives}{True\ Positives + False\ Negatives}. \tag{6.5}$$

$$FPR = \frac{False\ Positives}{False\ Positives + True\ Negatives}. \tag{6.6}$$

where $True\ Positives$ (respectively $True\ Negatives$) is the number of attacks (respectively normal connections) correctly classified. $False\ Positives$ is the number of normal connections incorrectly reported as attacks and $False\ Negatives$ is the number of attacks incorrectly reported as normal connections.

Terminal and function sets

The terminal set includes input variables, which are the 41 KDD-99 feature sets with eight randomly generated constants. The function set includes basic arithmetic, comparison and logical operators reaching 17 functions (Table 6.6).

Function (node) set	
Arithmetic operators:	$+, -, *, \%$
Comparison operators:	$<, >, <=, >=, =$
Logic operators:	AND, OR, NOT, NOR, NAND
Other:	NEGATE, IF (IF THEN ELSE), IFLEZE(IF <=0 THEN ELSE)
Terminal set	
KDD-99 Features	41
Ephemeral Random Constants	8 in $[-2, 2[$

Table 6.6. *Terminal and function sets for GP*

CGP parameters

The values of the CGP parameters used in this work are summarized in Table 6.7. They are obtained from a series of trials.

Parameter	Value
Population size	512
Number of generations	500
CGP nodes	300
Levels-back	3
Inputs/Outputs	49/1
Tournament size	4
Crossover/Mutation probability	0.9/0.04
Fitness	Minimize classification errors

Table 6.7. *CGP parameters.*

6.6.3. *Results*

Experiments are performed on an Intel i7-4810MQ, 2.8GHz, 64 bits workstation. To compare the performance of the implemented sampling methods, height measures are recorded for each of them through 21 runs: the best and the mean values for the accuracy (equation 6.4), TPR (equation 6.5) and FPR (equation 6.6) metrics as well as the run time. These measures are illustrated by Figure 6.15 for CGP without sampling (full dataset: FSS), CGP extended with RSS and CGP extended with DSS. They are obtained with the test set presented in Table 6.5.

With accuracy and TPR values exceeding 90%, CGP was able to efficiently handle this classification problem and obtain competitive results compared to other approaches [SAB 03]. Otherwise, experimental results show that introducing a sampling method to the GP system improves its performance when training from large datasets. Both the implemented sampling methods (RSS and DSS) have better results than the conventional CGP (FSS) according to the Accuracy, FPR and TPR metrics. The computational cost is also lower, i.e. CGP+RSS and CGP+DSS are much faster than CGP+FSS without sampling. However, CGP+DSS seems rather more robust given that, eventually, all data entries were selected to participate in several different subsets thanks to the “age component” of the technique (section 6.6.1).

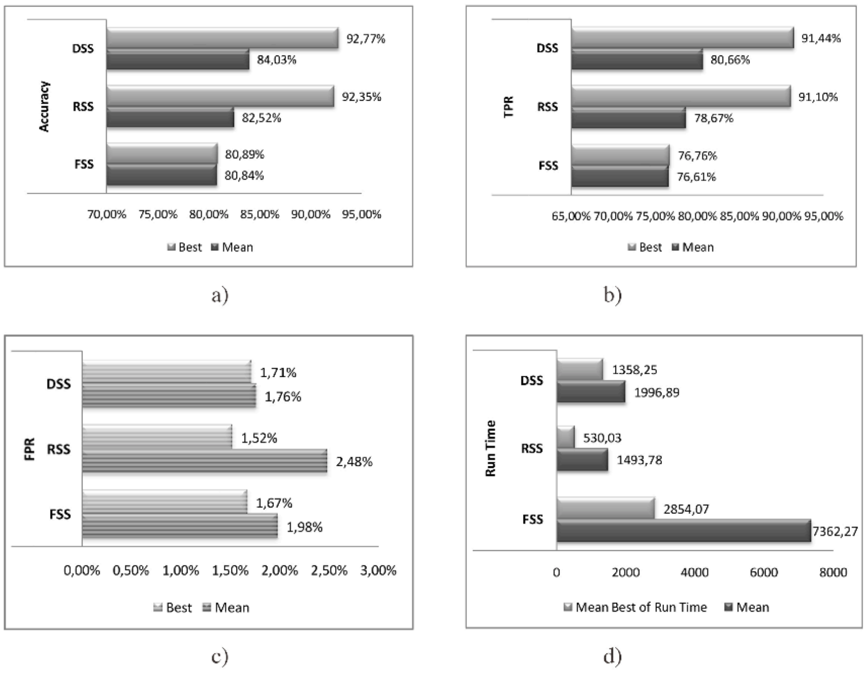


Figure 6.15. *Performance of CGP+FSS, CGP+RSS, CGP+DSS for the intrusion classification problem according to the three metrics: a) Accuracy, b) TPR, c) FPR d) run times in seconds*

6.7. Conclusion

Genetic Programming algorithms are evolutionary algorithms that attempt to solve the problem: *"How can a computer program be made to do what needs to be done without being told exactly how to do it?"* [KIN 94]. As such, they can share the same parental and environmental selection as other evolutionary algorithms. However, they use specific encoding and variation operators to represent and modify computer programs. Thus, they are at the intersection of two larger disciplines: *Optimizatio*n and *Artificial intelligence*, more specifically: *Machine Learning*.

Thanks to Koza's first book *"Genetic Programming: On the Programming of Computers by Means of Natural selection"* which shows how GP is able to breed computer programs capable of solving, or approximately solving, a wide variety of problems from a wide variety of fields, the scientific community has shown interest in these algorithms. Researchers have proposed further extensions and new representations to improve the GP systems. GP variants have proven to be able to solve complex and advanced problems in machine intelligence. GP is now able to compete with other advanced methods in artificial intelligence such as Neural Networks.

For further readings about Genetic Programming technology and applications, we suggest the series of Koza's books [KOZ 92, KOZ 94, KOZ 99a, KOZ 06, KOZ 10b] which give details of GP backgrounds and theoretical foundations, guidelines and examples for GP applications and their contribution to the automated invention machine. For more details about of linear and graphic GP representations, we suggest the following books: "Linear Genetic Programming" by Brameier and Banzhaf [BRA 07], "Gene Expression Programming: mathematical modeling by an artificial intelligence" by Ferreira [FER 06] and "Cartesian Genetic Programming" by Miller [MIL 11].

Bibliography

[ACK 87] ACKLEY D.H., *A Connectionist Machine for Genetic Hillclimbing*, Kluwer, 1987.

[AGU 04] AGUIRRE A.H., RIONDA S.B., COELLO COELLO C.A. *et al.*, "Handling constraints using multiobjective optimization concepts", *International Journal for Numerical Methods in Engineering*, vol. 59, no. 15, pp. 1989–2017, 2004.

[AMB 91] AMBATI B.K., AMBATI J., MOKHTAR M.M., "Heuristic combinatorial optimization by simulated Darwinian evolution: a polynomial time algorithm for the traveling salesman problem", *Biological Cybernetics*, vol. 65, no. 1, pp. 31–35, 1991.

[ANG 96] ANGELINE P.J., "Two self-adaptive crossover operators for genetic programming", in ANGELINE P.J., KINNEAR JR. K.E. (eds.), *Advances in Genetic Programming 2*, MIT Press, Cambridge, 1996.

[ANG 97] ANGELINE P.J., "An alternative to indexed memory for evolving programs with explicit state representations", *Genetic Programming*, vol. 97, pp. 423–430, 1997.

[AUG 05] AUGER A., HANSEN N., "A restart CMA evolution strategy with increasing population size", *2005 IEEE International Congress on Evolutionary Computation (CEC'05)*, pp. 1769–1776, 2005.

[BAC 91] BACK T., HOFFMEISTER F., SCHWEFEL H.-P., "A survey of evolution strategies", *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 2–9, 1991.

[BAD 11] BADER J., ZITZLER E., "Hype: an algorithm for fast hypervolume-based many-objective optimization", *Evolutionary Computation*, vol. 19, no. 1, pp. 45–76, March 2011.

[BAK 87] BAKER J.E., "Reducing bias and inefficiency in the selection algorithm", *Proceedings of the 2nd International Conference on Genetic Algorithms*, pp. 14–21, 1987.

[BAN 93] BANZHAF W., "Genetic programming for pedestrians", *Proceedings of the 5th International Conference on Genetic Algorithms (ICGA-93)*, p. 628, University of Illinois at Urbana-Champaign, 17–21 July 1993.

[BEA 96] BEASLEY J.E., CHU P.C., "A genetic algorithm for the set covering problem", *European Journal of Operational Research*, vol. 94, no. 2, pp. 392–404, 1996.

[BEN 00] BEN-HAMIDA S., SCHOENAUER M., "An adaptive algorithm for constrained optimization problems", *Proceedings of 6th Parallel Problem Solving From Nature (PPSN VI)*, pp. 529–538, Paris, France, September 2000.

[BEN 02] BEN-HAMIDA S., SCHOENAUER M., “ASCHEA: new results using adaptive segregational constraint handling”, *Proceedings of the Congress on Evolutionary Computation 2002 (CEC’2002)*, pp. 884–889, Piscataway, NJ, May 2002.

[BEU 07] BEUME N., NAUJOKS B., EMMERICH M., “SMS-EMOA: multiobjective selection based on dominated hypervolume”, *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, September 2007.

[BEY 92] BEYER H., “Some aspects of the evolution strategy for solving TSP-like optimization problems”, in MANNER R., MANDERICK B. (eds.), *Parallel Problem Solving from Nature 2*, Elsevier, 1992.

[BEY 01] BEYER H., *The Theory of Evolution Strategies, Natural Computing Series*, Springer, 2001.

[BEY 02] BEYER H., SCHWEFEL H., “Evolution strategies – a comprehensive introduction”, *Natural Computing*, vol. 1, no. 1, pp. 3–52, March 2002.

[BOO 89] BOOKER L.B., GOLDBERG D.E., HOLLAND J.H., “Classifier systems and genetic algorithms”, *Artificial Intelligence*, vol. 40, nos. 1–3, pp. 235–282, September 1989.

[BRA 07] BRAMEIER M.F., BANZHAF W., *Linear Genetic Programming*, Springer Science & Business Media, 2007.

[BRE 06] BREST J., GREINER S., BOSKOVIC B. *et al.*, “Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems”, *Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, December 2006.

[CAI 06] CAI Z., WANG Y., “A multiobjective optimization-based evolutionary algorithm for constrained optimization”, *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 658–675, December 2006.

[CAM 97] CAMPONOGARA E., TALUKDAR S.N., “A genetic algorithm for constrained and multiobjective optimization”, *Proceedings of the 3rd Nordic Workshop on Genetic Algorithms and their Applications*, pp. 49–61, 1997.

[CHA 98] CHAMBERS L.D., *Practical Handbook of Genetic Algorithms: Complex Coding Systems*, vol. 3, CRC Press, 1998.

[CHA 13] CHAUHAN H., KUMAR V., PUNDIR S. *et al.*, “A comparative study of classification techniques for intrusion detection”, *2013 International Symposium on Computational and Business Intelligence (ISCBI)*, pp. 40–43, 2013.

[CHE 99] CHENG R., GEN M., TSUJIMURA Y., “A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies”, *Computers & Industrial Engineering*, vol. 36, no. 2, pp. 343–364, 1999.

[CLE 02] CLERC M., KENNEDY J., “The particle swarm – explosion, stability, and convergence in a multidimensional complex space”, *Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, February 2002.

[CLE 06] CLERC M., “Stagnation Analysis in Particle Swarm Optimisation or What Happens When Nothing Happens”, available at: https://hal.archives-ouvertes.fr/hal-00122031, 2006.

[CLE 12] CLERC M., “Standard Particle Swarm Optimisation”, available at: https://hal.archives-ouvertes.fr/hal-00764996, 2012.

[CLE 07] CLEGG J., WALKER J.A., MILLER J.F., “A new crossover technique for Cartesian genetic programming”, *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, vol. 2, pp. 1580–1587, London, 2007.

[COE 99] COELLO C.A.C., “Self-adaptive penalties for GA-based optimization”, *Proceedings of the Congress on Evolutionary Computation 1999 (CEC’99)*, vol. 1, Piscataway, NJ, pp. 573–580, July 1999.

[COE 02a] COELLO C.A.C., “Theoretical and numerical constraint handling techniques used with evolutionary algorithms: a survey of the state of the art”, *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11–12, pp. 1245–1287, January 2002.

[COE 02b] COELLO C.A.C., MEZURA-MONTES E., “Handling constraints in genetic algorithms using dominance-based tournaments”, in PARMEE I. ed., *Proceedings of the Fifth International Conference on Adaptive Computing in Design and Manufacture (ACDM’2002)*, vol. 5, pp. 273–284, University of Exeter, Devon, April 2002.

[COE 02c] COELLO C.A.C., MONTES E.M., “Constraint-handling in genetic algorithms through the use of dominance-based tournament selection”, *Advanced Engineering Informatics*, vol. 16, no. 3, pp. 193–203, 2002.

[CRA 85] CRAMER N.L., “A representation for the adaptive generation of simple sequential programs”, *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 183–187, 1985.

[DAR 59] DARWIN C., *On the Origin of Species by Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life*, Murray, London, 1859.

[DAS 98] DAS I., DENNIS J.E., “Normal-boundary intersection: a new method for generating Pareto optimal points in multicriteria optimization problems”, *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 631–657, August 1998.

[DAS 16] DAS S., MULLICK S.S., SUGANTHAN P., “Recent advances in differential evolution – an updated survey”, *Swarm and Evolutionary Computation*, vol. 27, pp. 1–30, 2016.

[DAV 85] DAVIS L., “Job shop scheduling with genetic algorithms”, *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 136–140, Hillsdale, NJ, 1985.

[DAV 91] DAVIS L., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[DE 93a] DE JONG K.A., SARMA J., “Generation gaps revisited”, in WHITLEY L.D. (ed.), *Foundations of Genetic Algorithms 2*, Morgan Kaufmann, 1993.

[DE 93b] DE LA MAZA M., TIDOR B., “An analysis of selection procedures with particular attention paid to proportional and Boltzmann selection”, *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 124–131, 1993.

[DEB 89] DEB K., GOLDBERG D.E., “An investigation of niche and species formation in genetic function optimization”, *Proceedings of the 3rd International Conference on Genetic Algorithms*, San Francisco, CA, pp. 42–50, 1989.

[DEB 95] DEB K., AGRAWAL R.B., “Simulated binary crossover for continuous search space”, *Complex Systems*, vol. 9, pp. 115–148, 1995.

[DEB 00] DEB K., “An efficient constraint handling method for genetic algorithms”, *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2/4, pp. 311–338, 2000.

[DEB 01] DEB K., *Multi-objective Optimization using Evolutionary Algorithms*, John Wiley and Sons, 2001.

[DEB 02] DEB K., PRATAP A., AGARWAL S. *et al.*, “A fast and elitist multiobjective genetic algorithm: NSGA-II”, *Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.

[DEB 03] DEB K., MOHAN M., MISHRA S., A fast multi-objective evolutionary algorithm for finding well-spread Pareto-optimal solutions, Report no. KanGAL 2003002, Indian Institute of Technology Kanpur, 2003.

[DEB 14] DEB K., JAIN H., “An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints”, *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014.

[DEE 11] DEEP K., ADANE H.M., “New variations of order crossover for travelling salesman problem”, *International Journal of Combinatorial Optimization Problems and Informatics*, vol. 2, no. 1, p. 2, 2011.

[EBE 95] EBERHART R.C., KENNEDY J., “A new optimizer using particle swarm theory”, *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39–43, 1995.

[EBE 00] EBERHART R.C., SHI Y., “Comparing inertia weights and constriction factors in particle swarm optimization”, *Proceedings of the 2000 Congress on Evolutionary Computation*, pp. 84–88, Piscataway, NJ, 2000.

[EIB 11] EIBEN A.E., SMIT S.K., “Parameter tuning for configuring and analyzing evolutionary algorithms.”, *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 19–31, 2011.

[ELA 13] EL-ABD M., “Testing a particle swarm optimization and artificial bee colony hybrid algorithm on the CEC13 benchmarks”, *IEEE Congress on Evolutionary Computation*, IEEE, pp. 2215–2220, 2013.

[ELK 00] ELKAN C., “Results of the KDD’99 classifier learning”, *SIGKDD Explorations Newsletter*, vol. 1, no. 2, pp. 63–64, ACM, January 2000.

[ELS 14] ELSAYED S.M., SARKER R.A., ESSAM D.L., “A self-adaptive combined strategies algorithm for constrained optimization using differential evolution”, *Applied Mathematics and Computation*, vol. 241, pp. 267–282, 2014.

[ESH 93] ESHELMAN L.J., SCHAFFER J.D., “Real-coded genetic algorithms and interval-schemata”, *Proceedings of Foundations of Genetic Algorithms 2*, pp. 187–202, 1993.

[FAR 02] FARINA M., AMATO P., “On the optimal solution definition for many-criteria optimization problems”, *2002 Annual Meeting of the North American Fuzzy Information Processing Society Proceedings*, pp. 233–238, 2002.

[FAR 03] FARMANI R., WRIGHT J.A., “Self-adaptive fitness formulation for constrained optimization”, *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 445–455, October 2003.

[FER 06] FERREIRA C., *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*, vol. 21, Springer, 2006.

[FOG 66] FOGEL L.J., OWENS A.J., WALSH M.J., *Artificial Intelligence through Simulated Evolution*, Wiley, 1966.

[FOG 88] FOGEL D.B., “An evolutionary approach to the traveling salesman problem”, *Biological Cybernetics*, vol. 60, no. 2, pp. 139–144, 1988.

[FON 93] FONSECA C.M., FLEMING P.J., “Genetic algorithms for multiobjective optimization: formulation, discussion and generalization”, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 416–423, 1993.

[FON 95] FONSECA C.M., FLEMING P.J., “An overview of evolutionary algorithms in multiobjective optimization”, *Evolutionary Computation*, vol. 3, pp. 1–16, 1995.

[GAT 94] GATHERCOLE C., ROSS P., “Dynamic training subset selection for supervised learning in genetic programming”, *Parallel Problem Solving from Nature – PPSN III*, vol. 866, pp. 312–321, 1994.

[GEN 94] GEN M., TSUJIMURA Y., KUBOTA E., “Solving job-shop scheduling problems by genetic algorithm”, *1994 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 2, pp. 1577–1582, 1994.

[GEN 00] GEN M., CHENG R., *Genetic Algorithms and Engineering Optimization*, vol. 7, John Wiley & Sons, 2000.

[GOL 85] GOLDBERG D.E., LINGLE R., “Alleles, loci, and the traveling salesman problem”, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, vol. 154, pp. 154–159, Hillsdale, NJ, 1985.

[GOL 89] GOLDBERG D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[GOL 91] GOLDBERG D.E., DEB K., “A comparison of selection schemes used in genetic algorithms”, in RAWLINS G. (ed.), *Foundations of Genetic Algorithms*, Morgan Kaufmann, 1991.

[GRE 85] GREFENSTETTE J., GOPAL R., ROSMAITA B. *et al.*, “Genetic algorithms for the traveling salesman problem”, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pp. 160–168, New Jersey, NJ, 1985.

[GRE 87] GREFENSTETTE J.J., “Incorporating problem specific knowledge into genetic algorithms”, *Genetic Algorithms and Simulated Annealing*, vol. 4, pp. 42–60, 1987.

[GUO 15] GUO S.-M., TSAI J.S.H., YANG C.-C. *et al.*, “A self-optimization approach for L-SHADE incorporated with eigenvector-based crossover and successful-parent-selecting framework on CEC 2015 benchmark set”, *CEC*, pp. 1003–1010, 2015.

[HAD 97] HADJ-ALOUANE A.B., BEAN J.C., “A genetic algorithm for the multiple-choice integer program”, *Operations Research*, vol. 45, pp. 92–101, 1997.

[HAM 00] HAMIDA S.B., PETROWSKI A., “The need for improving the exploration operators for constrained optimization problems”, *Proceedings of the Congress on Evolutionary Computation 2000 (CEC’2000)*, vol. 2, pp. 1176–1183, Piscataway, NJ, July 2000.

[HAN 96] HANSEN N., OSTERMEIER A., “Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation”, *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 312–317, May 1996.

[HAN 04] HANSEN N., KERN S., “Evaluating the CMA evolution strategy on multimodal test functions”, *PPSN*, vol. 3242, pp. 282–291, 2004.

[HAN 06] HANSEN N., “The CMA evolution strategy: a comparing review”, in LOZANO J., LARRAAGA P., INZA I. *et al.* (eds.), *Towards a New Evolutionary Computation*, pp. 75–102, Springer, Berlin, 2006.

[HAN 09] HANSEN N., “Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed”, *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers* (GECCO ’09), pp. 2389–2396, New York, 2009.

[HAN 16] HANSEN N., “The CMA evolution strategy: a tutorial”, available at: https://hal.inria.fr/hal-01297037, 2016.

[HAR 06] HARTL D.L., CLARK A.G., *Principles of Population Genetics*, 4th ed., Sinauer Associates, 2006.

[HE 07] HE Q., WANG L., “A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization”, *Applied Mathematics and Computation*, vol. 186, no. 2, pp. 1407–1422, Elsevier, 2007.

[HEL 07] HELWIG S., WANKA R., “Particle swarm optimization in high-dimensional bounded search spaces”, *Proceedings of IEEE Swarm Intelligence Symposium (SIS)*, pp. 198–205, 2007.

[HMI 16a] HMIDA H., BEN HAMIDA S., BORGI A. *et al.*, “Hierarchical data topology based selection for large scale learning”, *Workshop on High Performance Big Data Computing (WHPBDC)*, 2016.

[HMI 16b] HMIDA H., HAMIDA S.B., BORGI A. *et al.*, “Sampling methods in genetic programming learners from large datasets: a comparative study”, *Advances in Big Data: Proceedings of the 2nd INNS Conference on Big Data*, pp. 50–60, Thessaloniki, Greece, 23–25 October 2016.

[HOL 75] HOLLAND J.H., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, University of Michigan Press, 1975.

[HOL 92] HOLLAND J.H., *Adaptation in Natural and Artificial Systems*, 2nd ed., MIT Press, 1992.

[HOM 94] HOMAIFAR A., LAI S.H.Y., QI X., “Constrained optimization via genetic algorithms”, *Simulation*, vol. 62, no. 4, pp. 242–254, 1994.

[HOR 94] HORN J., NAFPLIOTIS N., GOLDBERG D.E., “A niched Pareto genetic algorithm for multiobjective optimization”, *Proceedings of 1st IEEE Conference on Evolutionary Computation*, pp. 82–87, Piscataway, NJ, 1994.

[JOI 94] JOINES J., HOUCK C., “On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs”, *Proceedings of the First IEEE Conference on Evolutionary Computation*, Orlando, Florida, pp. 579–584, 1994.

[KAR 15] KARAFOTIAS G., HOOGENDOORN M., EIBEN A.E., "Parameter control in evolutionary algorithms: trends and challenges", *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 167–187, 2015.

[KEN 95] KENNEDY J., EBERHART R.C., "Particle swarm optimization", *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995.

[KHU 94] KHURI S., BÄCK T., HEITKÖTTER J., "The zero/one multiple knapsack problem and genetic algorithms", *Proceedings of the 1994 ACM Symposium on Applied Computing*, pp. 188–193, 1994.

[KIN 94] KINNEAR K.E., *Advances in Genetic Programming*, vol. 1, MIT Press, 1994.

[KNO 99] KNOWLES J., CORNE D., "The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation", *Proceedings of the Congress on Evolutionary Computation*, vol. 1, Washington, DC, pp. 98–105, June–September 1999.

[KNO 00] KNOWLES J.D., CORNE D.W., "Approximating the nondominated front using the Pareto archived evolution strategy", *Evolutionary Computation*, vol. 8, no. 2, pp. 149–172, MIT Press, June 2000.

[KNO 02] KNOWLES J., CORNE D., "On metrics for comparing nondominated sets", *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*, vol. 1, pp. 711–716, May 2002.

[KOZ 92] KOZA J.R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[KOZ 94] KOZA J.R., *Genetic Programming II: Automatic Discovery of Reusable Subprograms*, MIT Press, 1994.

[KOZ 99a] KOZA J.R., *Genetic Programming III: Darwinian Invention and Problem Solving*, vol. 3, Morgan Kaufmann, 1999.

[KOZ 99b] KOZIEL S., MICHALEWICZ Z., "Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization", *Evolutionary Computation*, vol. 7, no. 1, pp. 19–44, 1999.

[KOZ 06] KOZA J.R., KEANE M.A., STREETER M.J. *et al.*, *Genetic Programming IV: Routine Human-competitive Machine Intelligence*, vol. 5, Springer Science & Business Media, 2006.

[KOZ 10] KOZA J.R., "Human-competitive results produced by genetic programming", *Genetic Programming and Evolvable Machines*, vol. 11, no. 3–4, pp. 251–284, September 2010.

[KUM 06] KUMAR R., BANERJEE N., "Analysis of a multiobjective evolutionary algorithm on the 0–1 knapsack problem", *Theoretical Computer Science*, vol. 358, no. 1, pp. 104–120, 2006.

[LAN 00] LANGDON W., "Size fair and homologous tree crossovers for tree genetic programming", *Genetic Programming and Evolvable Machines*, vol. 1, no. 1–2, pp. 95–119, 2000.

[LAN 05] LANGDON W., BANZHAF W., "Repeated sequences in linear genetic programming genomes", *Complex Systems*, vol. 15, no. 4, pp. 285–306, 2005.

[LAN 12] LANGDON W., *Genetic Programming and Data Structures: Genetic Programming+Data Structures=Automatic Programming!*, vol. 1, Springer Science & Business Media, 2012.

[LAR 99] LARRAÑAGA P., KUIJPERS C.M.H., MURGA R.H. *et al.*, "Genetic algorithms for the travelling salesman problem: a review of representations and operators", *Artificial Intelligence Review*, vol. 13, no. 2, pp. 129–170, 1999.

[LAU 02] LAUMANNS M., THIELE L., DEB K. *et al.*, "Combining convergence and diversity in evolutionary multiobjective optimization", *Evolutionary Computation*, vol. 10, no. 3, pp. 263–282, MIT Press, September 2002.

[LEG 07] LEGUIZAMÓN G., COELLO C.A.C., "A boundary search based ACO algorithm coupled with stochastic ranking", *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007)*, pp. 165–172, Singapore, 25–28 September 2007.

[LER 95] LE-RICHE R.G., KNOPF-LENOIR C., HAFTKA R.T., "A segregated genetic algorithm for constrained structural optimization", *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95)*, San Mateo, CA, pp. 558–565, July 1995.

[LI 14] LI M., YANG S., LIU X., "Shift-based density estimation for Pareto-based algorithms in many-objective optimization", *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 348–365, 2014.

[LI 15] LI B., LI J., TANG K. *et al.*, "Many-objective evolutionary algorithms: a survey", *ACM Computing Surveys*, vol. 48, no. 1, pp. 13:1–13:35, September 2015.

[LIA 13] LIANG J.J., QU B.Y., SUGANTHAN P.N., Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization, Report, Zhengzhou University and Nanyang Technological University, 2013.

[LUK 97] LUKE S., SPECTOR L., "A comparison of crossover and mutation in genetic programming", *Genetic Programming*, vol. 97, pp. 240–248, 1997.

[LUK 98] LUKE S., SPECTOR L., "A revised comparison of crossover and mutation in genetic programming", *Genetic Programming*, vol. 98, no. 208–213, p. 55, Citeseer, 1998.

[MAL 10] MALLIPEDDI R., SUGANTHAN P.N., "Ensemble of constraint handling techniques", *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 4, pp. 561–579, August 2010.

[MES 03] MESSAC A., ISMAIL-YAHAYA A., MATTSON C., "The normalized normal constraint method for generating the Pareto frontier", *Structural and Multidisciplinary Optimization*, vol. 25, no. 2, pp. 86–98, Springer-Verlag, 2003.

[MEZ 09] MEZURA-MONTES E. ed., *Constraint-Handling in Evolutionary Optimization*, Springer, Berlin, Germany, 2009.

[MEZ 11] MEZURA-MONTES E., COELLO C.A.C., "Constraint-handling in nature-inspired numerical optimization: past, present and future", *Swarm and Evolutionary Computation*, vol. 1, no. 4, pp. 173–194, 2011.

[MIC 91] MICHALEWICZ Z., JANIKOW C.Z., "Handling constraints in genetic algorithms", in BELEW R.K., BOOKER L.B. (eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA-91)*, San Mateo, CA, pp. 151–157, 1991.

[MIC 92] MICHALEWICZ Z., *Genetic Algorithms+Data Structures=Evolution Programs*, Springer-Verlag Berlin, Heidelberg, 1992.

[MIC 94] MICHALEWICZ Z., ATTIA N.F., "Evolutionary optimization of constrained problems", *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pp. 98–108, 1994.

[MIC 95] MICHALEWICZ Z., NAZHIYATH G., "Genocop III: a co-evolutionary algorithm for numerical optimization with nonlinear constraints", *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pp. 647–651, Piscataway, NJ, 1995.

[MIC 96a] MICHALEWICZ Z., *Genetic Algorithms+Data Structures=Evolution Programs*, 3rd ed., Springer Verlag, 1996.

[MIC 96b] MICHALEWICZ Z., DASGUPTA D., RICHE R.L. *et al.*, "Evolutionary algorithms for constrained engineering problems", *Computers & Industrial Engineering Journal*, vol. 30, no. 4, pp. 851–870, September 1996.

[MIC 96c] MICHALEWICZ Z., SCHOENAUER M., "Evolutionary algorithms for constrained parameter optimization problems", *Evolutionary Computation*, vol. 4, no. 1, pp. 1–32, 1996.

[MIC 12] MICHALEWICZ Z., "Quo Vadis, evolutionary computation?", *Advances in Computational Intelligence: IEEE World Congress on Computational Intelligence* (WCCI 2012), Brisbane, Australia, 10–15 June 2012.

[MIL 99] MILLER J.F., "An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming approach", *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 2, pp. 1135–1142, Orlando, FL, 13–17 July 1999.

[MIL 00] MILLER J.F., THOMSON P., "Cartesian genetic programming", *Proceedings of the Third European Conference on Genetic Programming (EuroGP-2000)*, vol. 1802, pp. 121–132, Edinburgh, Scotland, 15–16 April 2000.

[MIL 11] MILLER J.F., *Cartesian Genetic Programming*, Springer, 2011.

[MON 95] MONTANA D.J., "Strongly typed genetic programming", *Evolutionary Computation*, vol. 3, no. 2, pp. 199–230, 1995.

[MOR 06] MORAGLIO A., TOGELIUS J., LUCAS S., "Product geometric crossover for the Sudoku puzzle", *2006 IEEE International Conference on Evolutionary Computation*, pp. 470–476, 2006.

[MOR 07] MORAGLIO A., KIM Y.-H., YOON Y. *et al.*, "Geometric crossovers for multiway graph partitioning", *Evolutionary Computation*, vol. 15, no. 4, pp. 445–474, 2007.

[MÜH 88] MÜHLENBEIN H., GORGES-SCHLEUTER M., KRÄMER O., "Evolution algorithms in combinatorial optimization", *Parallel Computing*, vol. 7, no. 1, pp. 65–85, 1988.

[MÜH 89] MÜHLENBEIN H., "Parallel genetic algorithms, population genetics and combinatorial optimization", *Workshop on Parallel Processing: Logic, Organization, and Technology*, Springer, pp. 398–406, 1989.

[MÜH 91] MÜHLENBEIN H., "Evolution in time and space – the parallel genetic algorithm", in *Foundations of Genetic Algorithms*, Morgan Kaufmann, 1991.

[MYU 98] MYUNG H., KIM J.-H., "Hybrid interior-lagrangian penalty based evolutionary optimization", *Proceedings of the 7th International Conference on Evolutionary Programming (EP98)*, vol. 1447, pp. 85–94, San Diego, CA, March 1998.

[NER 12] NERI F., COTTA C., MOSCATO P., *Handbook of Memetic Algorithms*, vol. 379, Springer, 2012.

[NOM 01] NOMURA T., SHIMOHARA K., "An analysis of two-parent recombinations for real-valued chromosomes in an infinite population", *Evolutionary Computation*, vol. 9, no. 3, pp. 283–308, MIT Press, 2001.

[NOR 94] NORDIN P., "A compiling genetic programming system that directly manipulates the machine code", in KINNEAR JR. K.E. (ed.), *Advances in Genetic Programming*, MIT Press, 1994.

[NOR 99] NORDIN P., BANZHAF W., FRANCONE F.D., "Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover", in SPECTOR L., LANGDON W.B., O'REILLY U.-M. *et al.* (eds.), *Advances in Genetic Programming 3*, MIT Press, Cambridge, MA, 1999.

[OLI 87] OLIVER I.M., SMITH D.J., HOLLAND J.R.C., "A study of permutation crossover operators on the traveling salesman problem", *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, Hillsdale, NJ, pp. 224–230, 1987.

[OLT 02] OLTEAN M., DUMITRESCU D., Multi expression programming, Report, Universitatea Babe-Bolyai, 2002.

[OLT 03] OLTEAN M., GROŞAN C., "Evolving evolutionary algorithms using multi expression programming", *European Conference on Artificial Life*, pp. 651–658, 2003.

[O'NE 01] O'NEILL M., RYAN C., "Grammatical evolution", *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[O'NE 03] O'NEILL M., RYAN C., *Grammatical Evolution*, Springer, 2003.

[ONO 97] ONO I., SATOH H., KOBAYASHI S., "A genetic algorithm based on sub-sequence exchange crossover and GT method for job-shop scheduling", *IEEJ Transactions on Electronics, Information and Systems*, vol. 117, no. 7, pp. 888–895, 1997.

[ONW 09] ONWUBOLU G.C., DAVENDRA D., *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*, vol. 175, Springer Science & Business Media, 2009.

[OSA 13a] OSABA E., CARBALLEDO R., DIAZ F. *et al.*, "New results in the suitability analysis of using blind crossover operators in genetic algorithms for solving routing problems", *Scientific Bulletin of the Politehnica University of Timioara, Romania Transactions on Automatic Control and Computer Science*, vol. 58no. 72, pp. 111–118, 2013.

[OSA 13b] OSABA E., CARBALLEDO R., DIAZ F. *et al.*, "Analysis of the suitability of using blind crossover operators in genetic algorithms for solving routing problems", *2013 IEEE 8th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pp. 17–22, 2013.

[OSA 14] OSABA E., DIAZ F., ONIEVA E. *et al.*, "AMCPA: a population metaheuristic with adaptive crossover probability and multi-crossover mechanism for solving combinatorial optimization problems", *International Journal of Artificial Intelligence*, vol. 12, no. 2, pp. 1–23, 2014.

[OSM 12] OSMAN I.H., KELLY J.P., *Meta-heuristics: Theory and Applications*, Springer Science & Business Media, 2012.

[PAN 05] PANAIT L., LUKE S., "Cooperative multi-agent learning: the state of the art", *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, November 2005.

[PAR 94] PARMEE I.C., PURCHASE G., “The development of a directed genetic search technique for heavily constrained design spaces”, *Adaptive Computing in Engineering Design and Control-’94*, University of Plymouth, pp. 97–102, 1994.

[PAR 06] PARK T., RYU K.R., “A dual-population genetic algorithm for balanced exploration and exploitation”, *Computational Intelligence*, pp. 90–95, 2006.

[PER 94] PERKIS T., “Stack-based genetic programming”, *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, vol. 1, pp. 148–153, Orlando, FL, 27–29 1994.

[POL 97] POLI R. et al., “Evolution of graph-like programs with parallel distributed genetic programming”, *ICGA*, pp. 346–353, 1997.

[POL 99] POLI R., “Parallel distributed genetic programming”, in CORNE D., DORIGO M., GLOVER F. (eds.), *New Ideas in Optimization*, McGraw-Hill, Maidenhead, 1999.

[POL 07] POLI R., KENNEDY J., BLACKWELL T., “Particle swarm optimization”, *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[POL 08] POLI R., LANGDON W., MCPHEE N.F. *et al.*, *A Field Guide to Genetic Programming*, Lulu. com, 2008.

[POW 93] POWELL D., SKOLNICK M.M., “Using genetic algorithms in engineering design optimization with non-linear constraints”, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, San Mateo, CA, pp. 424–431, July 1993.

[PRI 05] PRICE K.V., STORN R.M., LAMPINEN J.A., *Differential Evolution: A Practical Approach to Global Optimization*, Springer-Verlag, 2005.

[PUL 13] PULJIĆ K., MANGER R., “Comparison of eight evolutionary crossover operators for the vehicle routing problem”, *Mathematical Communications*, vol. 18, no. 2, pp. 359–375, 2013.

[RAD 90] RADCLIFFE N., Genetic Neural Networks on MIMD Computers, PhD Thesis, University of Edinburgh, 1990.

[RAY 00] RAY T., KANG T., CHYE S.K., “An evolutionary algorithm for constrained optimization”, *Genetic and Evolutionary Computation Conference*, pp. 771–777, 2000.

[REC 65] RECHENBERG I., *Cybernetic Solution Path of an Experimental Problem*, Royal Aircraft Establishment Library Translation, 1965.

[REC 73] RECHENBERG I., *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog, Stuttgart, 1973.

[ROG 99] ROGERS A., PRÜGEL-BENNETT A., “Genetic drift in genetic algorithm selection schemes”, *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 298–303, 1999.

[ROW 04] ROWE J., WHITLEY D., BARBULESCU L. *et al.*, “Properties of gray and binary representations”, *Evolutionary Computation*, vol. 12, no. 1, pp. 47–76, 2004.

[RUD 92] RUDOLPH G., “On correlated mutations in evolution strategies”, *Parallel Problem Solving from Nature*, Elsevier, pp. 105–114, 1992.

[RUN 00] RUNARSSON T., YAO X., “Stochastic ranking for constrained evolutionary optimization”, *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 284–294, 2000.

[RUN 06] RUNARSSON T., "Approximate evolution strategy using stochastic ranking", in YEN G.G., LUCAS S.M., FOGEL G. *et al.* (eds.), *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pp. 745–752, Vancouver, BC, 16–21 July 2006.

[RYA 98] RYAN C., COLLINS J.J., NEILL M.O., "Grammatical evolution: evolving programs for an arbitrary language", *Proceedings of the First European Workshop on Genetic Programming*, vol. 1391, pp. 83–96, 1998.

[SAB 03] SABHNANI M., SERPEN G., "Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection context", *MLMTA*, pp. 209–215, 2003.

[SCH 81] SCHWEFEL H.-P., *Numerical Optimization of Computer Models*, Wiley, 1981.

[SCH 85] SCHAFFER J.D., "Multiple objective optimization with vector evaluated genetic algorithms", *International Conference on Genetic Algorithms*, pp. 93–100, 1985.

[SCH 93] SCHOENAUER M., XANTHAKIS S., "Constrained GA optimization", *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, San Mateo, CA, pp. 573–580, July 1993.

[SCH 95] SCHOENAUER M., LAMY B., JOUVE F., Identification of mechanical behaviour by genetic programming part II: energy formulation, Report, Ecole Polytechnique, France, 1995.

[SCH 96a] SCHOENAUER M., MICHALEWICZ Z., "Evolutionary computation at the edge of feasibility", *Proceedings of the Fourth Conference on Parallel Problem Solving from Nature (PPSN IV)*, Berlin, Germany, pp. 245–254, September 1996.

[SCH 96b] SCHOENAUER M., SEBAG M., JOUVE F. *et al.*, "Evolutionary identification of macro-mechanical models", in ANGELINE P.J., KINNEAR JR. K.E. (eds.), *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, 1996.

[SCH 97] SCHOENAUER M., MICHALEWICZ Z., "Boundary operators for constrained parameter optimization problems", *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA-97)*, San Francisco, California, pp. 322–329, July 1997.

[SCH 98] SCHOENAUER M., MICHALEWICZ Z., "Sphere operators and their applicability for constrained optimization problems", *Proceedings of the 7th International Conference on Evolutionary Programming (EP98)*, San Diego, CA, vol. 1447, pp. 241–250, March 1998.

[SIN 08] SINGH H.K., ISAACS A., RAY T. *et al.*, "Infeasibility driven evolutionary algorithm (IDEA) for engineering design optimization", *Australasian Conference on Artificial Intelligence*, pp. 104–115, 2008.

[SIN 09] SINGH H.K., ISAACS A., NGUYEN T.T., *et al*, "Performance of infeasibility driven evolutionary algorithm (IDEA) on constrained dynamic single objective optimization problems", *2009 IEEE Congress on Evolutionary Computation (CEC'2009)*, pp. 3127–3134, Trondheim, Norway, May 2009.

[SMI 93] SMITH A.E., TATE D.M., "Genetic optimization using a penalty function", *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, San Mateo, CA, pp. 499–503, July 1993.

[SRI 94] SRINIVAS N., DEB K., "Multiobjective function optimization using nondominated sorting genetic algorithms", *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1994.

[STA 91] STARKWEATHER T., MCDANIEL S., MATHIAS K.E. *et al.*, "A comparison of genetic sequencing operators", *ICGA*, pp. 69–76, 1991.

[STO 97] STORN R., PRICE K., “Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces”, *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[SUR 95] SURRY P.D., RADCLIFFE N.J., BOYD I.D., “A multi-objective approach to constrained optimisation of gas supply networks: the COMOGA method”, *Evolutionary Computing, AISB Workshop, Selected Papers*, Sheffield, UK, vol. 993, pp. 166–180, April 1995.

[SYS 89] SYSWERDA G., “Uniform crossover in genetic algorithms”, *Proceedings of the 3rd International Conference on Genetic Algorithms*, San Francisco, CA, pp. 2–9, 1989.

[SYS 91] SYSWERDA G., “Schedule optimization using genetic algorithms”, in DAVIS L. (ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[TAK 04] TAKAHAMA T., SAKAI S., “Constrained optimization by α constrained genetic algorithm (αGA)”, *Systems and Computers in Japan*, vol. 35, no. 5, pp. 11–22, 2004.

[TAK 06] TAKAHAMA T., SAKAI S., IWANE N., “Solving nonlinear constrained optimization problems by the ε constrained differential evolution”, *2006 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2322–2327, 2006.

[TAN 13a] TANABE R., FUKUNAGA A., “Evaluating the performance of SHADE on CEC 2013 benchmark problems”, *IEEE Congress on Evolutionary Computation*, IEEE, pp. 1952–1959, 2013.

[TAN 13b] TANABE R., FUKUNAGA A., “Success-history based parameter adaptation for differential evolution”, *IEEE Congress on Evolutionary Computation*, pp. 71–78, 2013.

[TAN 14] TANABE R., FUKUNAGA A., “Improving the search performance of SHADE using linear population size reduction”, *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014*, Beijing, China, pp. 1658–1665, 6–11 July 2014.

[TAO 98] TAO G., MICHALEWICZ Z., “Inver-over operator for the TSP”, *Parallel Problem Solving from Nature – PPSN V: 5th International Conference*, Amsterdam, Netherlands, 27–30 September, pp. 803–812, 1998.

[TAV 09] TAVALLAEE M., BAGHERI E., LU W. *et al.*, “A detailed analysis of the KDD CUP 99 data set”, *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009.

[TEL 95] TELLER A., VELOSO M.M., PADO: learning tree structured algorithms for orchestration into an object recognition system, Report no. CMU-CS-95-101, Carnegie-Mellon University, Computer Science, Pittsburgh, 1995.

[TEL 96] TELLER A., “Evolving programmers: the co-evolution of intelligent recombination operators”, in ANGELINE P.J., KINNEAR JR. K.E. (eds.), *Advances in Genetic Programming 2*, MIT Press, 1996.

[TES 06] TESSEMA B., YEN G.G., “A self adaptative penalty function based algorithm for constrained optimization”, *2006 IEEE Congress on Evolutionary Computation (CEC’2006)*, Vancouver, BC, pp. 950–957, July 2006.

[UMB 15] UMBARKAR A., JOSHI M., SHETH P., “Dual population genetic algorithm for solving constrained optimization problems”, *International Journal of Intelligent Systems and Applications*, vol. 7, no. 2, p. 34, 2015.

[URB 09] URBANOWICZ R.J., MOORE J.H., “Learning classifier systems: a complete introduction, review, and roadmap”, *Journal of Artificial Evolution and Applications*, vol. 2009, pp. 1:1–1:25, January 2009.

[VEL 00] VAN VELDHUIZEN D.A., LAMONT G.B., “Multiobjective optimization with messy genetic algorithms”, *SAC (1)’00*, pp. 470–476, 2000.

[WAS 01] WASIEWICZ P., MULAWKA J.J., “Molecular genetic programming”, *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, vol. 2, no. 5, pp. 106–113, 2001.

[WHI 89] WHITLEY L.D., STARKWEATHER T., FUQUAY D., “Scheduling problems and traveling salesmen: the genetic edge recombination operator”, *ICGA*, vol. 89, pp. 133–40, 1989.

[WHI 09] WHITE D.R., POULDING S., “A rigorous evaluation of crossover and mutation in genetic programming”, *European Conference on Genetic Programming*, pp. 220–231, 2009.

[WHI 12] WHILE R.L., BRADSTREET L., BARONE L., “A fast way of calculating exact hypervolumes”, *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 1, pp. 86–95, 2012.

[WIL 08] WILSON G., BANZHAF W., “A comparison of Cartesian genetic programming and linear genetic programming”, *European Conference on Genetic Programming*, pp. 182–193, 2008.

[WOL 97] WOLPERT D.H., MACREADY W.G., “No free lunch theorems for optimization”, *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[YOO 12] YOON Y., KIM Y.-H., MORAGLIO A. *et al.*, “Quotient geometric crossovers and redundant encodings”, *Theoretical Computer Science*, vol. 425, pp. 4–16, 2012.

[ZAH 02] ZAHARIE D., “Critical values for the control parameters of differential evolution algorithms”, *Proceedings of MENDEL*, vol. 2, p. 6267, 2002.

[ZAM 13] ZAMBRANO-BIGIARINI M., CLERC M., ROJAS R., “Standard Particle Swarm Optimisation 2011 at CEC-2013: a baseline for future PSO improvements”, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2013)*, pp. 2337–2344, Cancun, Mexico, 20–23 June 2013, 2013.

[ZHA 03] ZHANG W.-J., XIE X.-F., “DEPSO: hybrid particle swarm with differential evolution operator”, *IEEE International Conference on Systems Man and Cybernetics*, vol. 4, pp. 3816–3821, 2003.

[ZHA 07] ZHANG Q., LI H., “MOEA/D: a multiobjective evolutionary algorithm based on decomposition”, *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[ZHA 09] ZHANG J., SANDERSON A.C., “JADE: adaptive differential evolution with optional external archive”, *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 945–958, 2009.

[ZHA 15] ZHANG Y., WANG S., JI G., “A comprehensive survey on particle swarm optimization algorithm and its applications”, *Mathematical Problems in Engineering*, vol. 2015, 2015.

[ZIT 99] ZITZLER E., THIELE L., "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach", *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[ZIT 02] ZITZLER E., LAUMANNS M., THIELE L., "SPEA2: improving the strength Pareto evolutionary algorithm for multiobjective optimization", *Evolutionary Methods for Design, Optimisation, and Control*, Barcelona, Spain, pp. 95–100, 2002.

[ZIT 03] ZITZLER E., THIELE L., LAUMANNS M. *et al.*, "Performance assessment of multiobjective optimizers: an analysis and review", *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

[ZYD 01] ZYDALLIS J.B., VAN VELDHUIZEN D.A., LAMONT G.B., "A statistical comparison of multiobjective evolutionary algorithms including the MOMGA-II", *EMO'01*, pp. 226–240, 2001.

Index