POLITECNICO DI MILANO

Facoltà di Ingegneria Industriale

Corso di Laurea in
Ingegneria Energetica

Differential Evolution for the optimization of complex technological systems: application to the oil&gas and nuclear industries.

Relatore: Prof. Enrico Zio

Co-relatore1: Ing. Francesco VERRE

Co-relatore2: Ing. Alberto CASAROTTI

Tesi di Laurea di:

Giorgio VIADANA Matr. 734550

Anno Accademico 2009 - 2010.

Index

Listo of figures

Listo of tables

Sommario

L’ottimizzazione di processi complessi, come quelli industriali, spesso risulta essere un obiettivo difficilmente perseguibile senza specifiche competenze ed esperienze acquisite sul campo. La difficoltà di tali sistemi deriva principalmente dalla complessa forma che essi possono avere, dalle diverse interazioni che legano proprietà, operatività e performance unite all’incertezza che ne deriva. Il mondo dell’Oil&Gas e quello del Nucleare sono due tra le branche dell’industria dove più si sente il bisogno di strumenti in grado di ottimizzare, simulare e dare risposte a quesiti di difficile risoluzione tramite una semplice analisi del sistema. Gli algoritmi evolutivi, come gli Algoritmi Genetici o il Differential Evolution, possono dare notevoli miglioramenti nella definizione di gestione di asset produttivi o di impianti nucleari.
Tale tesi espone in maniera completa le varie tecniche del Differential Evolution sviluppate in questi anni, analizzando la bontà di tali tecniche, le situazioni in cui possono essere utilizzati e la sensitività dei loro parametri. Inoltre, due casi studio su problemi reali del mondo dell’industria sono presentati e risolti grazie all’applicazione di questo potente strumento di ottimizzazione.

Abstract

The optimization of complex processes, such as industrial systems, often turns out to be a goal difficult to pursue without specific expertise and experience in the field. The difficulties of such systems are derived primarily from the complex form they may have, the different interactions that bind ownership, operation and performance combined with the uncertainty it brings. The Oil & Gas world and the Nuclear industry are two of the branches where tools to optimize, simulate and provide answers to questions difficult to resolve by a simple analysis are necessary. Evolutionary algorithms, such as Genetic Algorithms or Differential Evolution, can make significant improvements in the definition of asset management or production of nuclear plants.
The thesis sets out comprehensively the various techniques of Differential Evolution developed in recent years, analyzing the goodness of such techniques, the situations in which they can be used and the sensitivity of their parameters. In addition, two case studies on real problems of industry are presented and resolved through the application of this powerful optimization tool.

Keywords: Evolutionary Algorithms, Genetic Algorithms, Differential Evolution, Single-objective optimization, Multi-objective optimization

Chapter 1

Optimization

Optimization is the process of adjusting the variables or parameters of a system or process to achieve the minimum or maximum of some given objectives. Several ways could be taken to find the optimum; anyway its definition is unique. For simplicity, from here on we will speak in terms of minimization.

Say the system of interest has P properties

$$p_k; \quad k \in \{1, 2, \ldots, P\} \tag{1.1}$$

and C constraints

$$h_m; \quad m \in \{1, 2, \ldots, C\} \tag{1.2}$$

which are dependent on n real variables

$$x_j; \quad j \in \{1, 2, \ldots, n\} \tag{1.3}$$

Usually the variables have a domain defined by the upper and lower bounds

$$x_j \in \left[x_j^L, x_j^U \right] \tag{1.4}$$

and the whole of these variables form a solution inside the domain D

$$\underline{x} = (x_1, x_2, \ldots, x_n): \quad \underline{x} \in D \subseteq \mathbb{R}^n \tag{1.5}$$

They affect the p_k properties and h_m constraints, so

$$\begin{aligned} p_k &= f_k(\underline{x}) \\ h_m &= f_m(\underline{x}) \end{aligned} \tag{1.6}$$

The constraints can be equality constraint, like

$$h_m = 0 \tag{1.7}$$

or inequality constraint, like

$$h_m \leq 0 \tag{1.8}$$

They define the feasible region $\Omega \subseteq D$ within finding the optimized set of variables

$$\underline{x}^* = \left(x_1^*, x_2^*, \ldots, x_n^* \right) \tag{1.9}$$

that satisfies the optimization problem.

When the optimization goal is to minimize a single property, the task is to find

$$\underline{x}^* \Big| f_k\left(\underline{x}^*\right) < f_k\left(\underline{x}\right) \quad \forall \underline{x} \in \Omega \tag{1.10}$$

where f_k is referred to the property p_k to minimize.

The problem must be reformulated in the case of multi-objective optimization: the aim is to generate a list of non-dominated solutions, called Pareto list, within which each solution cannot be said to be better of another one considering all the objective functions. The solution of this kind of problems generates a so called Pareto frontier which represents the whole of non-inferior (or equally good) sets of variables that satisfy optimization and constraints.

In that case the optimization target is a vector of objective functions

$$\underline{F}\left(\underline{x}\right) = \left(f_1\left(\underline{x}\right), f_2\left(\underline{x}\right), \ldots, f_P\left(\underline{x}\right) \right) \tag{1.11}$$

The problem can be formulated defining these two operators, $\not\equiv$ and $\preceq$, related to the concept of non-dominance [1].

Assuming two candidate vector solutions, $\underline{x}$ and $\underline{y}$, we say that they are different

$$\underline{x} \not\equiv \underline{y} \quad if \quad \exists \; x_j \in \underline{x},\, y_j \in \underline{y} \,|\, x_j \neq y_j \tag{1.12}$$

And that $\underline{y}$ is dominated by $\underline{x}$

$$\underline{x} \preceq \underline{y} \quad if \quad \forall \; x_j \in \underline{x},\; y_j \in \underline{y} \,|\, x_j \leq y_j \; and \; \underline{x} \not\equiv \underline{y} \tag{1.13}$$

An efficient, non-inferior/Pareto-optimal solution is a vector

$$\underline{x}^* \in \Omega \; if \; \nexists \underline{x} \in \Omega | \underline{F}(\underline{x}) \preceq \underline{F}\left(\underline{x}^*\right) \tag{1.14}$$

The difference between single and multi-objective optimization is relevant: in the first case the finding of the global optimum for a single property leads to obtain just one solution, one set of variables that satisfies the condition of dominance in the domain *D* for the required objective. On the contrary, when optimization means finding a vector of equally good solutions, the optimization task becomes hard and some degree of complexity is introduced.

The multi-objective optimization problem requires the finding of many configurations that satisfy the concept of arrays' superiority. This situation is quite frequent in real optimization, especially for complex industrial systems: many conflicting targets to optimize, constraints to satisfy and the difficulty to homogenize their different quantities, such as reliability, costs, pollution impacts and health consequences, impose a multi-objective optimization.

The finding of only one solution does not satisfy the target of the problem, since other non-dominated configurations exist. In practice, after the definition of the Pareto frontier by multi-objective optimization, only one solution could be applied to the real case. The final choice of the solution to realize is left to the human decision, affected by other considerations (e.g. economic, politic, side effects or environmental considerations not included into the optimization). Nevertheless, the definition of the Pareto front must be as clear as possible, to give to the user all the information available for the decision.

Chapter 2

Introduction on Evolutionary Algorithms for optimization

Chapter 1 describes the optimization task for single-objective (SO) or multi-objective (MO) optimization. An important aspect of the optimization is the number of variable involved into the definition of the problem and the number of constraints introduced.

If the number of these variables is small and the objective functions are differentiable and linear, as for the constraints, a typical gradient method's optimization works well and fast. If the objective functions depend from several parameters and non-linear constraints, or the objective functions are not differentiable, a direct search approach is really useful. The direct search methods belong to the class of optimization that do not compute derivatives. Algorithms like Nelder and Mead simplex method [2], parallel direct search algorithm (PDS) [3] or Simulated Annealing [4] are examples of powerful methods.

Anyhow, classic optimization and direct search methods have the risk to be trapped by local minima, since they find only one solution every search. Local perturbation of the solutions is one interesting attempt made to escape from local minima, but for problems with high complexity and high multimodality this method fails (e.g.: functions 6, 15, 19, 22 and 23 of Appendix A).

Evolutionary Algorithms (EAs) are an attractive alternative to traditional methods of optimization, especially for problems with high complexity, high number of constraints or high dimensionality.

They are a class of stochastic algorithms for optimization inspired by the Darwin's theory of evolution: a population of potential solutions is launched in the search and it is able to adapt to its surrounding environment; it evolves, ruled by heuristics, in a way that those solutions best satisfying the optimization objectives are more likely to contribute to the future generations of solutions (the survival of fittest); the fitness definition is made by a fitness or objective function that describes the features of any solution. EAs need only information about the environment and about the fitness function itself, without any further information about continuity or differentiability: in fact, they lay into the class of direct search method, but the revolutionary idea is their global searching from a population of solutions rather than one single solution.

Thanks to their population-based approach, they have high capabilities to escape from inconvenient situations like local optima, since their simultaneous

probabilistic manipulation of several solutions; farther, this approach is extremely suitable for MO, because of the need of several solutions as result. Furthermore, their evolution concept gives to the process flexibility respect to the different problem's nature.

These features are extremely hopeful for the optimization, both in SO and MO optimization.

To overcome this difficulty of a MO optimization, a non-dominated comparison is usually adopted [1], but in these years several different strategies for MO in EA are been proposed [5-11].

For EAs, each individual is represented by a specific combination of independent variables, or, mathematically speaking, by a *n*-dimensional vector (called also chromosome) contained inside the domain *D* that is a hypothetical solution of the optimization problem:

$$\underline{x}_i = \left(x_{1i}, x_{2i}, \ldots, x_{ni} \right): \quad \underline{x}_i \in D \subseteq \mathbb{R}^n \tag{2.1}$$

Each solution vector's fitness needs to be evaluated and the corresponding values are used to probabilistically rule the constitution of the successive generation.

Since their population approach, we need to define the population *S* with *NP* chromosomes of the *G* generation

$$\underline{x}_{i,G}; \quad i \in \{1, 2, \ldots, NP\} \tag{2.2}$$

$$S_G = \left\{ \underline{x}_{1,G}, \underline{x}_{2,G} \ldots, \underline{x}_{NP,G} \right\} \tag{2.3}$$

The generation is the reference of the population *S* in a specific evolution time. Obviously, the first generation has index $G = 1$.

Heuristic rules, different for each family of EAs, are applied to this set of chromosome in order to find the global optimum. The population S_G is altered by these rules, its solutions are discarded if fittest ones are found and a new population, S_{G+1}, undergoes another time to the same procedure. In the long runs, this repetitive process allows the attainment of the optimal solution.

Proved the reliability of EAs in many artificial or real cases [12-17], the progresses obtained by these optimization techniques have inspired a number of alternatives; the two most widespread evolutionary techniques are Genetic Algorithms (GAs) and Differential Evolution (DE).

GA and DE generate offspring combining the chromosomes, generation by generation, and select by different rules solutionss to carry on the next generation.

Chapter 3

Genetic Algorithms: the ancestors of Differential Evolution

Genetic Algoritms (GAs) were firstly defined as optimization methods by Holland [18]. GAs are a particular class of EAs, and their functioning is inspired by the rules of the natural selection; furthermore, the procedures for recombination and generation of solutions resemble the principles of genetic, so, many terms in their definitions are borrowed by biology, coherently redefined to fit the algorithm contest.

3.1 Generalities

As the other EAs, GAs are characterized by a global searching based on a population approach.

The chromosome's variables are usually represented in binary coding, but, theoretically, any alphabet could be used [1].

To each variable is assigned a gene (see Figure 2.1). The length of each gene depends on the accuracy of the encoded variable. The combination of the *n*-genes is called chromosome, and its representation in any code is called genotype; in binary coding, the individual is characterized by a unique string of 0s and 1s.

Processing the information contained in the binary string, the fitness of any individual could be evaluated by the objective function: the genotype is decodified into real numbers, the control factors, one for each gene, defining the phenotype. The objective function (or fitness function) takes as input the phenotype and it renders the fitness. This value is then used as comparison for selection between individual in the population.

Defined a range $[x_j^L, x_j^U]$ for any j^{th} variable and assigned the number of bits nb_j for any gene, the relation between the control factor and its binary representation β is:

$$x_j = x_j^L + \beta \frac{x_j^U - x_j^L}{2^{nb_j}} \tag{3.1}$$

The values x_j^L, x_j^U and nb_j are called phenotyping parameters of the gene.

Figure 3.1 represents this concept on a problem with three variables involved: the number of genes is three, and, thanks to the coding/encoding procedure, the control factors x_1, x_2 and x_3 are obtained from the bit string. The resulting fitness is assigned to the chromosome.

This process is repeated for the entire population.

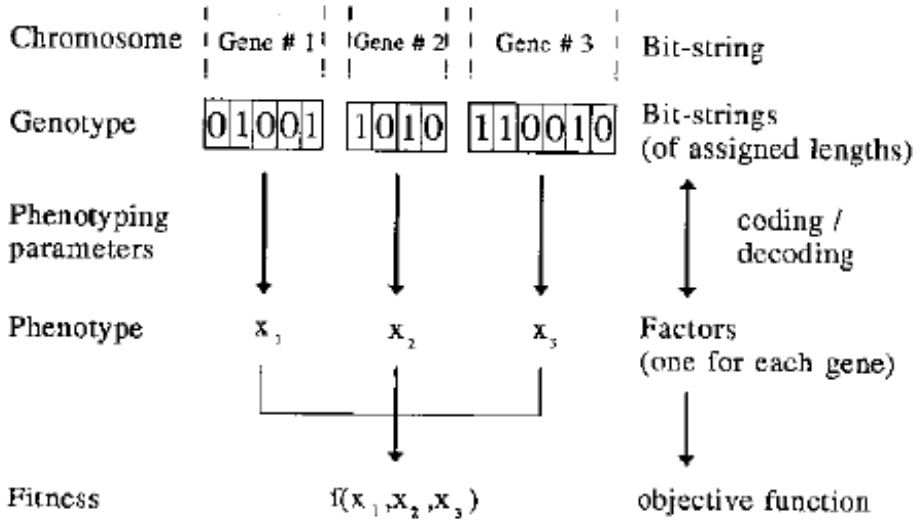


Figure 3.1 The binary encoding of the variable for GA [19]

GAs use specific operations in order to evolve the population: the main purposes of this evolution are the exploration of the search domain space and the consequent attainment of the global optimum of the system.

The GA search is performed by constructing a sequence of populations of chromosomes, the individuals of each population being the children of those of the previous population and the parents of those of the successive population. The initial population is generated by randomly sampling the bits of all the strings. At each step, the new population is then obtained by manipulating the strings of the old population in order to arrive at a new population hopefully characterized by increased mean fitness. This sequence continues until a termination criterion is reached. As for the natural selection, the string manipulation consists in selecting and mating pairs of chromosomes in order to groom chromosomes of the next population. This is done by repeatedly performing on the strings the four fundamental operations of *selection*, *crossover*, *replacement* and *mutation*, all based on random sampling: the parents' selection step determines the individuals which participate in the reproduction phase; reproduction itself allows the exchange of already existing genes whereas mutation introduces new genetic material; the substitution

defines the individuals for the next population. This way of proceeding enables to efficiently arrive at optimal or near-optimal solutions.

The classical GA steps are:

1. creation of a initial population of *NP* potential solutions to the problem and evaluation of their fitnesses;
2. selection of pairs of individuals as parents for reproduction;
3. crossover of the parents, with generation of two children;
4. evaluation of the children fitnesses;
5. replacement in the population with some rule, so as to maintain *NP* constant;
6. genetic mutation.
7. control for the stopping criteria, if some criterion is met stop, else go to step 2

When the children's fitnesses are evaluated, a replacement is made inside the population between parents and children and the population is dynamically updated. The new population is ranked by fitness criterion: in the long runs the best individuals will have a greater probability to be selected for mating, transmitting their genes to the children; these children have high chances to have good fitnesses, since they inherit good properties by their parents.

An important feature of the population is its genetic diversity: if the population is small, the scarcity of genetic information may provoke premature stagnation of the evolution, since the low possibility to exchange genetic material. However, if the population is too large, the overabundance of genetic material can lead a clustering of the population around local optima, decreasing the abilities of the reproduction process; then, the offspring fitness could be poor, because of lacking of good properties of either of the parents. Furthermore, the management of large population may be expensive in terms of computation time, with a high percentage of useless genetic material's processing. So, the population size, usually a used defined parameter, should not be too large or too small.

To avoid the premature stagnation or clustering, fresh genetic material is inserted by genetic mutation inside the population.

3.2 Genetic Algorithm operations

As deduced from this brief description, GA uses four operations to allow evolution: *selection*, *crossover*, *replacement* and *mutation*.

The first procedure performed by GA is the *selection* of parents for reproduction: the choice of the parents is one of the most important aspects, since it affects the goodness of the offspring.

Several variants exist, everyone with strengths and weaknesses: the choice from these variants is often affected by the problem nature and by the other choices made for the algorithm behaviour. They usually use fitness information that influence the selection; this device derives from the concept of the natural selection: individuals with high fitnesses has more probability to survive in the environment and to transmit their good properties to the progeny. The same approach is applied in GA.

The most used selection rules for mating are [19]:

- *Standard Roulette Selection*: the cumulative sum of the fitnesses of the individuals in the population is computed and normalized to sum to unity. A temporary population is generated by random sampling individuals, one at a time with replacement, from this cumulative sum. Then, the parents for mating are taken from this population. This procedure allows to the fittest individual in the population to be selected for this temporary population; by so doing, the mean fitness for the next population has good probabilities to be larger.
- *Hybrid Roulette Selection*: one disadvantage of the previous selection procedure is the fast lost on diversity for the next populations, since their mean fitnesses are fairly dispersed around the mean. In this procedure, after the normalization of the sum of the fitnesses, this cumulative term is multiplied by the population size: the integer part of this product is the number of individuals in the temporary population taken as they are from the current population. The remainder is selected with the Standard Roulette Selection. The permanence of good individuals is favoured, but the diversity could decline.
- *Random Selection and Mating*: the two parents are randomly selected over the entire population. This selection does not give any advantage to good individuals respect to the worse, with the possibility to destroy any achieved improvement.
- *Fit-fit selection and Mating*: after a ranking based on fitnesses, two parents are selected consequently from their rank. On the average, this procedure is highly conservative and the weakest individuals are soon eliminated. This method could provoke premature stagnation to local optima if the population size is not sufficiently high.
- *Fit-weak Selection and Mating*: the population is ranked as for the previous procedure, but each individual is paired with its symmetrical in the ranking. This practice improves the diversity but the improvements are small during the evolution.

Crossover is the main operator for GA: its main purpose is to mix the properties of different individuals, opportunely chosen.

During the generation of the offspring, the c*rossover* concept is used to reduce the search space to promising regions and at the same time to allow the

proceeding of the genome (the input variables) of a specific parent combined with the genome of a second parent using different strategies, like single or multi-site crossing.

In each pair of individual, chosen for mating, the corresponding genes are divided into two portions (one-site crossover) by a separation in the same position in both genes. Then, the first portions of the genes are exchanged. Two new chromosomes, the children, are produced, thanks to the combination of genetic material.

Figure 3.2 shows a single-site crossover.

A variation of this procedure consists on performing crossover only with an assigned probability p_c: a random number is generated by uniform distribution, $R \sim U(0,1)$, and the crossover is performed if $R < p_c$. otherwise, the two children are copies of the parents.

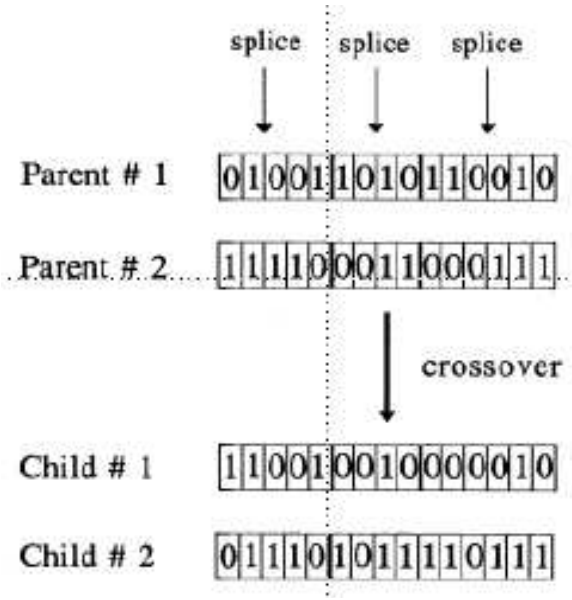


Figure 3.2, The single-site crossover operation for GA [19]

After the children generation and evaluation of their fitnesses, the *replacement* process mimics the survival of the fittest, allowing directly or indirectly the best solutions to continue the evolution: from the four chromosomes (two parents plus two children) two of them are selected to continue the evolution. The simplest recipe consists in the children replace the parents. Anyway, in GA many types of selection are proposed; this choice influences the entirely evolution process in terms of convergence speed and robustness, often connected with the diversity. The alternative procedures apply the *replacement* to selecting the chromosomes for substitution from the entire population; the most common are [19]:

- Fittest Individuals: the fittest two individuals from the group of four involved in and generated by crossover (two parents and the consecutive two children) replace the parents in the next generation; this procedure should not be used when the parent selection is too greedy and it does not select weak individuals (e.g. the Standard Roulette Selection).
- Weakest Individuals: the children just created replace the two weakest individuals in the population: this procedure could provoke premature stagnation, so it is recommended only if the population size is sufficiently high.
- Random Replacement: the children replace two randomly selected individual from the population; no fitness criterion is adopted in this procedure, so the attainment of the best is a task leaved to an efficient reproduction phase. This procedure works well in small population, and it allows deep search on the domain space.

The choice of one of the previous two operators (*crossover* and *replacement*) is affected by the other one: a correct interaction between these two is essential for the success of the search.

At the end, to increase the outcome of a GA, also random perturbations, *mutations*, can be introduced to avoid the possibility to be trapped in a local minimum: a defined percentage of the population is randomly mutated in order to insert new genetic material inside the population. The mutation is typically a flipping between two random bits or a random change from actual value to the opposite one. The mutation is performed on the basis of assigned mutation probability for any single bit (usually this value is quite small, like 10^{-3}).

When these operations are terminated, a control for stop is executed; the stopping criteria could be based on mean fitness of the solutions in the population, on best chromosome fitness, on weakest chromosome fitness or when a maximum number of generations is reached.

The option for GA that deserves attention is its encoding: GA in its basic version works manipulating a string containing 0s and 1s and altering or mixing the binary coding of different individuals with the purpose to generate fittest children. With the binary version of GA, the mating concept between parents is easily deducible.

Anyway, GA works also with real-coded variables, loosing in mating concept. The need of this encoding change arises from the quantization limitations of the binary one: when the variables are quantized, the binary GA fits nicely, but when the problem becomes continuous or the required precision becomes high, the floating-point representation is more appropriate. Then the chromosome does not have the semblances of a long string of 0s and 1s, as the chromosome, but becomes a mathematic entity of a vector. Also the evolutionary operations need some modification: the crossover concept is maintained, even in single or multi-cutting version, but a blending method remedy [20] is introduced. This method is simply a linear combination between two parents in order to enhance

the perturbations into the population, since the classic crossover applied to a vector is a merely interchanging of variables: when the dimensionality of the problem is small, this reproduction operation alone fails completely. The blending method is applied to each variable of the chromosome array:

$$\begin{aligned} x_{c_1,j} &= \alpha \cdot x_{p_1,j} + (1-\alpha) \cdot x_{p_2,j} \\ x_{c_2,j} &= (1-\alpha) \cdot x_{p_1,j} + \alpha \cdot x_{p_2,j} \end{aligned} \qquad (3.2)$$

$$\forall\, j \in \{1,2,\ldots,n\}$$

Where the subscript $c_{1\text{-}2}$ stays for *child* and $p_{1\text{-}2}$ stays for *parent*. The blending parameter α is taken from the range [0,1]. The other features, like parents selection, replacement and mutation are the same as for binary encoding.

3.3 Multi-objective optimization with Genetic Algorithms

When the optimization is SO optimization, the previous evolutionary operators use a simple comparison between the fitnesses (only one property of the system is the optimization target) of the individuals to select the parents or the individuals to be discarded.

When tackling a multi-objective problem by GAs, the various approaches to fitness definition may be distinguished into three categories [5] [18]:

- Aggregation methods combine the multiple objectives of the optimization into a scalar fitness function that is used to evaluate the goodness of a solution; an example is represented by the weighted-sum approach [1] [11], in which the fitness of solution is computed by the following weighted sum of the individual optimization objectives:

$$f_{weight}(\underline{x}) = \sum_{k=1}^{P} w_k \cdot f_k(\underline{x}) \qquad (3.3)$$

  where the arbitrary constant weights w_k, k=1,2,…,P satisfy the following relation:

$$w_k \in [0,1] \; and \sum_{k=1}^{P} w_k = 1 \qquad (3.4)$$

  The optimization of a single fitness function, combination of the objectives has the advantage of producing a single compromise solution, requiring no further selection by the decision maker. However, if the solution were found a posteriori not acceptable as a good compromise of the decision maker preferences, tuning of the aggregating weights may be required, followed by new runs of the optimizer, until a suitable solution is found.

- Population-based non-Pareto approaches are able to evolve multiple non-dominated solutions concurrently in a single simulation run: for instance, sub-populations of the next generation are reproduced from the current population separately for each of the objectives; then, the overall population at each generation is formed by merging and shuffling the sub-populations. The downside of this method is that it achieves a population of individuals that perform well for each objective separately, with no consideration given to trade-offs among them.
- In typical implementations of Pareto-based methods [1], the chromosomes of a population are ranked according to the Pareto dominance criterion applied to the fitnesses. With reference to the non-domination ranking, firstly, all non-dominated individuals are identified and rank 1 is assigned to them. Then, these solutions are virtually removed from the population and the next set of non-dominated individuals are identified and assigned rank 2; this process continues until every solution in the population has been ranked. Every solution belonging to the same rank class is Pareto-equivalent to any other of the same class and has the same probability of the others to be selected as a parent for the mating. Figure 3.3 shows an example of ranking for a set of solutions.

  During the optimization search, an archive of solution vectors, each one constituted by a non-dominated chromosome and by the corresponding fitnesses, representing the dynamic Pareto optimality set can be recorded and updated. This procedure also allows implementation of elitism in the genetic algorithm: in this work, every individual in the archive (or a pre-established number of individuals) is chosen once as a parent in each generation to guarantee a better propagation of the genetic code of non-dominated solutions and a more efficient evolution of the population towards Pareto optimality.

  At the end of the search procedure the result of the optimization is constituted by the archive itself which hopefully gives the Pareto-optimal set.

  The performance of a Pareto-based MOGA depends largely on its ability to maintain genetic diversity through the generations so as to arrive at a population of individuals which uniformly represent the real non-dominated solutions of the Pareto set, [11]. This can be achieved by resorting to niching techniques such as sharing [11].

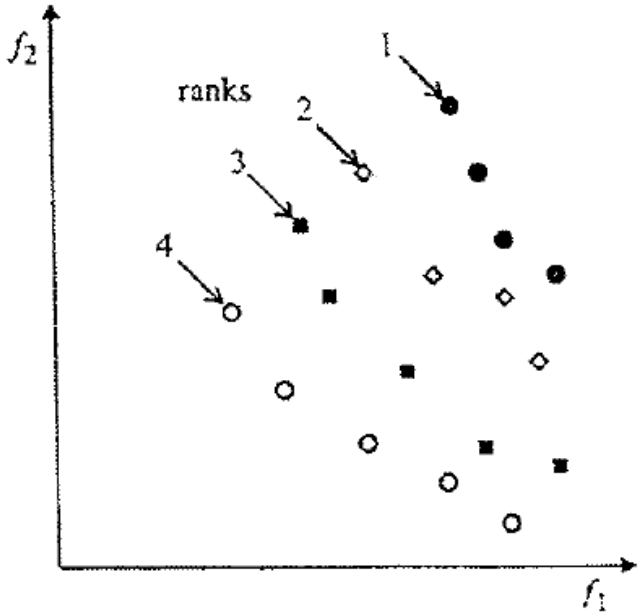


Figure 3.2, Example of population ranking for a maximization problem.

Chapter 4

Differential Evolution

DE arose from the Price's attempts to solve the Chebychev polynomial fitting problem posed to him by Storn [21]. It works similarly as GA, since both are EAs: it applies evolution operations on the individuals of the population in order to perturb them by transmission of good properties and find the global optimum of the system. One difference with respect to GA is that DE is specifically built for optimization over continuous spaces and does so based on a floating-point representation. The evolutionary operations are suited for such representation of the chromosome, and constitute the main improvement of DE with respect to GA, even in case of real-coded variables. The analogies between DE and real-coded GA are several, but the shrewdnesses adopted by these new operations are the strengths of the DE technique.

4.1 Basics

DE uses three heuristic operators as evolution strategies: they are called mutation, crossover and selection.

The revolutionary idea of DE is the perturbation to the current population. GA uses crossover between two parents for the generation of new solutions; these children inherit portion of genetic material from the parents. This mixing of properties is the main alteration in GA that perturbs the population. For DE, thanks to its real-coding, the representation of each individual is made by a vector instead a string of bits as in binary-encoding for GA. Then, the heuristics thought for DE are chosen with a view of vector operators.

The alteration for reproduction in DE, called mutation, is obtained adding to an individual the weighted difference between other two individuals randomly selected from the population.

This scheme is the original one proposed in [21]: for each vector xi,G in the population, called target vector, a noisy vector vi is generated randomly choosing three mutually different vector indices $r_1, r_2, r_3 \in \{1, 2, \ldots, NP\}$ with $i \notin \{r_1, r_2, r_3\}$:

$$\underline{v}_{i,G} = \underline{x}_{r_1,G} + F \cdot \left(\underline{x}_{r_2,G} - \underline{x}_{r_3,G} \right) \tag{4.1}$$

where the weighting (or scaling) factor $F \in (0,2]$ is a user-defined parameter, maintained constant during the optimization.

Figure 4.1 reports graphically the vectorial operation: to the vector $\underline{x}_{r1}$ the weighted difference between the vector $\underline{x}_{r2}$ and $\underline{x}_{r3}$ is added, to create the noisy vector $\underline{v}_i$. The difference between the vectors $\underline{x}_{r2}$ and $\underline{x}_{r3}$ is scaled by the factor F.

This linear combination between three solutions of the population is one of the revolutionary features of DE: using the weighted difference to perturb the population, the entire generation process becomes self-organized, because the step-length for the perturbation is mainly affected by the progress state of the evolution.

Through the evolution, the search space contracts or expands if the direction taken by the algorithm is correct or wrong, so the random step-length is self-adapted in every dimension accordingly with the dependence of the variable.

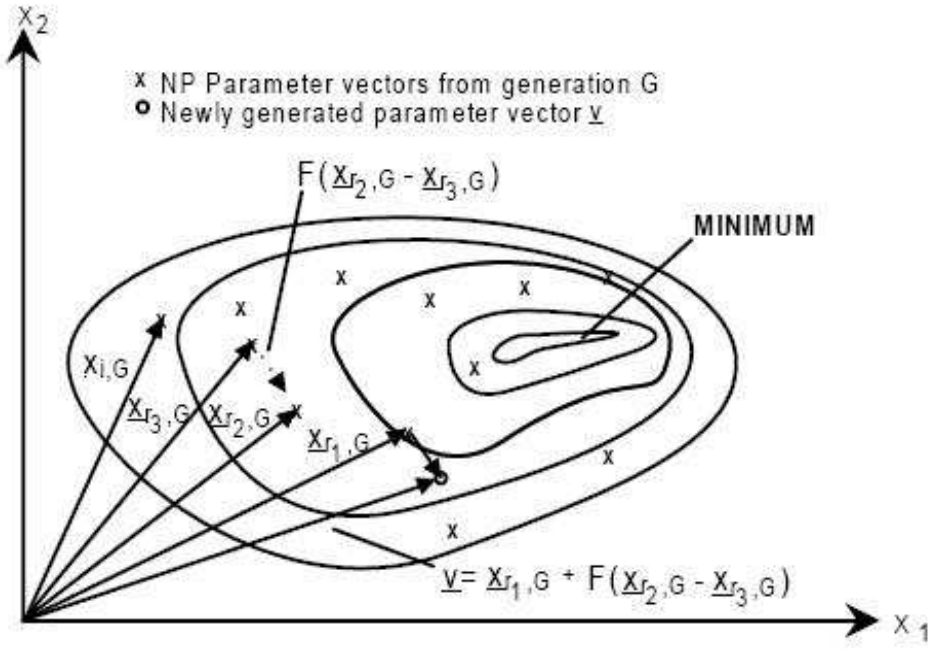


Figure 4.3, Noisy vector generation by mutation [22]

After *mutation*, the noisy vector is not directly compared with the target vector, but it is further modified by the *crossover* process, in which the noisy and target vector are mixed with some rule to create the trial vector $\underline{u}_i$, which inherits from them different pieces of chromosome. The *crossover* operator contributes to maintain the diversity inside the potential perturbed population, shuffling old and new information. This increases the probability to maintain some good property from the target vector, and avoids drastic changes during

generation of new solutions. The role of the *crossover* in DE has a secondary relevance compared to GA.

Due to the chromosome vectorial representation, the *crossover* operator for DE is applied to each element of the array: each variable of the noisy vector and the target vector has the possibility to be part of the trial vector, entering the final fight for the survival.

The most common crossover type adopted is the binomial approach: the trial vector is built by a modified Bernoulli trial rule (4.2), gauged by the control parameter $CR \in (0,1]$, which influences the probability for a noisy vector's chromosomes to be selected for the mutation process.

$$u_{ji,G} = \begin{cases} v_{ji,G} & if\ U(0,1] \le CR\ \ or\ \ j = irand(NP) \\ x_{ji,G} & if\ U(0,1] > CR\ \ and\ \ j \ne irand(NP) \end{cases} \qquad (4.2)$$

$$\forall j \in \{1, 2, \ldots, n\}$$

where $U(0,1]$ denotes the uniform continuous random value $\in (0,1]$, whereas $irand(NP)$ is a uniform discrete random number from the set $\{1,2,\ldots,NP\}$.

This ruling applied to the Bernoulli trials guarantees the inheriting of at least one component from the noisy vector in the trial vector even if the crossover rate *CR* is set to zero.

The binomial crossover operator acts on every "gene", represented by a variable, without any dependence between two neighbours, as for classic GA crossover. This one could be compared with a multi-site GA crossover affects by probability.

A relevant difference with GA is that in the DE *crossover* procedure a chromosome of the current population and one just generated, the noisy vector, are mixed, rather than two individuals of the population.

The resulting trial vector

$$\underline{u}_{i,G} = \left(u_{1i,G}, u_{2i,G}, \ldots, u_{ni,G}\right) \qquad (4.3)$$

inherits portions of noisy vector and from the target vector, as regulated by the parameter *CR*.

Figure 4.3 shows the principle underlying the binomial crossover process: the condition of the Bernoulli trial is met only for the variables' index 3, 4 and 6; the trial vector is then inherits the variables 1, 2, 5 and 7 from the target vector and the variables 3, 4 and 6 from the noisy one.

An alternative crossover scheme, the exponential crossover (see Section 4.2.2 of this chapter), was proposed by Storn and Price, [21]. This second crossover type works as a double-site crossover, allowing the interchanging of consequent

genes. This procedure shows less success and a more difficult setting. For this reason, binomial crossover is the most used crossover type.

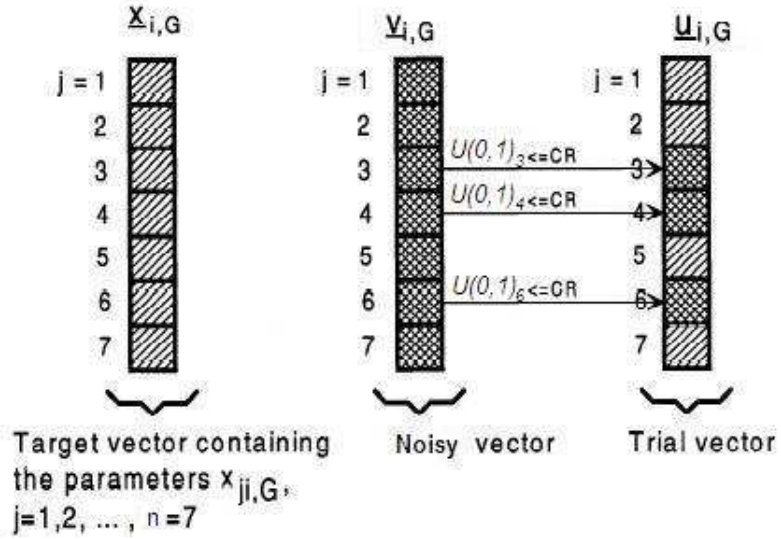


Figure 4, Binomial crossover for DE [22]

The trial vector obtained then enters the *selection* process where it is compared with the target vector $\underline{x}_{i,G}$ that is partially its parent, according with the crossover rule. During the selection process, the population S_G is modified by substitution.

Referring to a SO minimization, if the trial vector's fitness is less than the target vector's fitness, the latter will be a member of the next generation, replacing the target vector in the set S_{G+1} and the trial vector is discarded:

$$\underline{x}_{i,G+1} = \begin{cases} \underline{u}_{i,G} & \text{if } f_k\left(\underline{u}_{i,G}\right) < f_k\left(\underline{x}_{i,G}\right) \\ \underline{x}_{i,G} & \text{otherwise} \end{cases} \tag{4.4}$$

$$\forall i \in \{1, 2, \ldots, NP\}$$

The *selection* criterion in DE is greedy and quite different from the classical replacement criterion of GA: for sure the next generation will be better or at least equal of the previous generation.

The evolution for DE follows these steps:

1. creation of a initial population of *NP* potential solutions to the problem and evaluation of their fitnesses;
2. for each solution of the population (target vector) selection of three chromosomes for reproduction;
3. for each target vector, creation of a noisy vector using the mutation process;
4. creation of a trial vector mixing target and noisy vector;
5. comparison between each target vector and its related trial and eventual replacement;
6. control for the stopping criteria: if some criterion is met, then stop, else go to step 2.

The stopping criteria adoptable are the same as for GA (see Chapter 3).

This resulting EA, DE, shows robustness, higher convergence speed than GA and even better accuracy thanks to its greedy and well-chosen operators.

Like in GA the three operators must be balanced to allow the evolution and at the same time the exploration in the search space, but the convergence for DE is usually higher because the setting is less critical, thanks to the self-organization of the step-length, granting robustness to the strategy.

The key parameters of control for the basic DE presented are:

- *NP* population size
- *F* scaling factor
- *CR* crossover rate

Even if DE is more robust and suitable than GA, it has high possibility for improvement, especially for the convergence rate: since its basic structure enables sophistications, many strategies, concerning its operators, had been proposed with considerable successes, opening at the same time, in a parallel way, the problem of the control parameters setting that, as we shall see, it can be solved by a time-consuming and problem-dependant tuning or by adaptive/self-adaptive approach.

The initial population of the evolution process usually is composed by values distributed with random uniformity between the pre-specified lower x_j^L and upper x_j^U initial parameter bounds if they are specified, defining the domain *D*; so each variable of each individual is initialized as follow:

$$x_{ji,G} = x_j^L + U(0,1)\cdot\left(x_j^U - x_j^L\right) \tag{4.5}$$

$$\forall j \in \{1,2,\ldots,n\}, \forall i \in \{1,2,\ldots,NP\}, G = 1$$

This practice is beneficial because the exploration in the early phase, when the algorithm doesn't have particular preferential direction, is not unbalanced toward some region. Moreover, it is useful in the case the feasible region is coincident with the domain $\Omega = D$. In the case some solution is generated outside *D*, some repair rule is utilized (Section 4.3).

If the domain is not pre-defined, it is necessary define an initial region from which the algorithm can start.

Another possibility is starting from a previous solution, defining a range or giving the variance and mean around which generate initial individual if a Gauss distribution is desired. Of course, this initialization is used only in a forced optimization around a particular region or after a previous estimation.

An example of DE implementation is reported for Matlab.

```
%    Evolutionary Algorithm : Differential Evolution (DE)
%    Type of optimization: single-objective minimization
%
%---------------------------------------------------------------------
%                                parameters
%---------------------------------------------------------------------
NP=30;            % population number
CR=0.5;           % crossover frequency
F=0.5;            % scaling factor
MAXGEN=500;       % maximum number of generation
eps_alg=1e-4;     % difference limit between fmax and fmin
%---------------------------------------------------------------------
                              function options
%---------------------------------------------------------------------
low=-5.12;        % lower and upper bounds arrays
up=5.12;
dim=10;           % dimensionality
%---------------------------------------------------------------------
%                            initializations
%---------------------------------------------------------------------
k=1;       % generation index
matrix=zeros(NP,dim);          % matrix for the individuals
trial=zeros(NP,dim);           % trial vector
fitness=zeros(NP,1);           % fitness function
fitness_trial=zeros(NP,1);     % fitness of the trial vectors

% initial population uniformly distributed inside the domain
matrix=rand(NP,dim)*(up-low)+low;
%---------------------------------------------------------------------
%                              first evaluation
%---------------------------------------------------------------------
for i=1:NP
     fitness(i)=objectivefunction(matrix(i,:));
 end
f_max=max(fitness);
f_min=min(fitness);
delta=abs(f_max-f_min);
```

```
%-------------------------------------------------------------------------
%                                    code
%-------------------------------------------------------------------------
while (k<MAXGEN)&&(delta>=eps_alg)

    % ---------------------- %
    %        mutation
    %           &
    %      crossovering
    % ---------------------- %
    for i=1:NP
        r1=i;
        r2=i;
        r3=i;
        while (r1==i)
            r1=randi(NP);
        end
        while (r2==i)||(r2==r1)
            r2=randi(NP);
        end
        while (r3==i)||(r3==r1)||(r3==r2)
            r3=randi(NP);
        end

        p_add=randi(dim);

        for n=1:dim
           p=rand;
           if (p<=CR)||(p_add==n)
               trial(n)=matrix(r1,n)+F*(matrix(r2,n)-matrix(r3,n));
           else
               trial(n)=matrix(i,n);
           end
        end
    end
    % ---------------------- %
    %       evaluation
    % ---------------------- %

    for i=1:NP
           fitness_trial(i)=objectivefunction(trial(i,:));
    end

    % ---------------------- %
    %       selection
    % ---------------------- %
    for i=1:NP
        if (fitness_trial(i)<fitness(i)) matrix(i,:)=trial(i,:);
            fitness(i)=fitness_trial(i);
        end
    end

    % ---------------------- %
    %     best individual
    % ---------------------- %
    f_max=max(fitness);
    [f_min, i_best]=min(fitness);
    delta=abs(f_max-f_min);
```

```
    k=k+1;
end

bestindividual=matrix(i_best,:);
bestfitness=fmin;
```

4.2 Variants and sophistications

In the previous section we discussed about the main idea under DE, based on the easy rules of *mutation*, *crossover* and *selection*. How they were implemented gives as result greater robustness and convergence speed in with respect to GA, but several modifications, sophistications and shrewdnesses could be introduced to this basic DE scheme, since it leaves substantial rooms of improvements in the operators, especially in *mutation* and *crossover* procedures.

The first modified strategies were proposed by the creators Storn & Price [22]: they introduced ten different variants combining different operator's types, allowing the essential flexibility to the promising algorithm.

The general notation they proposed is *DE/x/y/z*, where

- *x* indicates the methods for selecting parents that will be used in the mutation process
- *y* indicates the number of vector differences used to perturb the base chromosome during the mutation process
- *z* indicates the crossover mechanism

The most widely used strategy is the one previously described, called *DE/rand/1/bin*, which uses random selection, one vector difference and binomial crossover.

Their attempt was the first in a series of modifications and improvements that lost the initial notation, treated in this section by operator.

4.2.1 Mutation options

Starting from the Storn & Price variants, in this section are described all the known mutation options presented in literature, with their strengths and weaknesses.

Storn & Price variants

The number of perturbations, pointed by the number of weighted differences *y*, is usually one or two, but in fact many more linear combinations of weighted differences could be introduced. The selection of $2 \cdot y$ vectors inside the population is generally made by a random sampling; the limitation is that the

vectors must be mutually different as well as for the base vector. This condition implies the minimum number for the population: it must be greater or equal to $2 \cdot y+1$ (anyway *NP* is usually greater than this value). Each vector difference is scaled by a F_y scaling factor, but in the classical formulation they are fixed and at the same value. Increasing the amount of vector differences is not a pursued practice because the random selection of the indices nullifies the expected amount of perturbation induced. An example of 2 vector differences perturbation with the same scaling factor is:

$$\underline{v}_{i,G} = \underline{x}_{basis,G} + F \cdot \left(\underline{x}_{r_1,G} - \underline{x}_{r_2,G} + \underline{x}_{r_3,G} - \underline{x}_{r_4,G} \right) \tag{4.6}$$

The basis vector depends on the selecting method used.

There are three selecting methods *x*. The following formulas show the case with one vector difference.

1. *rand*, proposed in the original DE version [21], plans to choose mutually different vectors from the current population, each with a uniform probability 1/*NP* (often the chosen indices must be different from the target vector index, in order to keep the crossover role, assigning probability 1/(*NP*-1) to each one), according with the number vector differences *y*: the number of vector's indices must be $2 \cdot y+1$. The first one is used as a base for the noisy vector, and the subsequent vectors for the differences. This technique avoids premature stagnation because of the random selection to the detriment of the convergence speed in some case.

$$\underline{v}_{i,G} = \underline{x}_{r_1,G} + F \cdot \left(\underline{x}_{r_2,G} - \underline{x}_{r_3,G} \right) \tag{4.7}$$

2. *best* uses as base vector in the mutation process the best solution in the current population. The other vectors for the differences are randomly chosen. This technique could enhance the speed of convergence but at the same time could destroy the necessary diversity to avoid stagnation; in the case the number of local optima is high, the algorithm fails with high probability because all the noisy vectors are direct toward a specific optimum. Only a lucky scaling factor setting, depending by the objective function's nature, could enable success. Otherwise for a non-critical and low-constrained objective function this method is interesting and appropriate.

$$\underline{v}_{i,G} = \underline{x}_{best,G} + F \cdot \left(\underline{x}_{r_1,G} - \underline{x}_{r_2,G} \right) \tag{4.8}$$

3. *current-to-best* uses as base vector the target vector, for this reason the selecting method contains the wording “current”. The perturbation is carefully driven toward the best of the actual generation and not around the best solution, since the basis vector is the target. Still a perturbation performed by a random vector difference exists: handling the two scaling factors involved the approach results less critical than the *best* configuration.

$$\underline{v}_{i,G} = \underline{x}_{i,G} + F_1 \cdot \left(\underline{x}_{best,G} - \underline{x}_{i,G}\right) + F_2 \cdot \left(\underline{x}_{r_1,G} - \underline{x}_{r_2,G}\right) \tag{4.9}$$

Figure 4.3 shows perfectly the non-critical approach: reducing F_1, the noisy vector could be generated not so close to the best of the generation. If F_1=1, the approach degenerates into the *best* configuration; otherwise, if F_1=0, the best solution is not taken into account and the basis vector is the current vector. This last situation is not advisable, because the crossover process becomes quite useless if not coupled with a high F_2: the crossover mixes the target vector with the noisy vector; then, if the basis for the latter is the target vector and the amount of perturbation doesn’t allow high exploration, far from the current population, the convergence speed could be very slow.

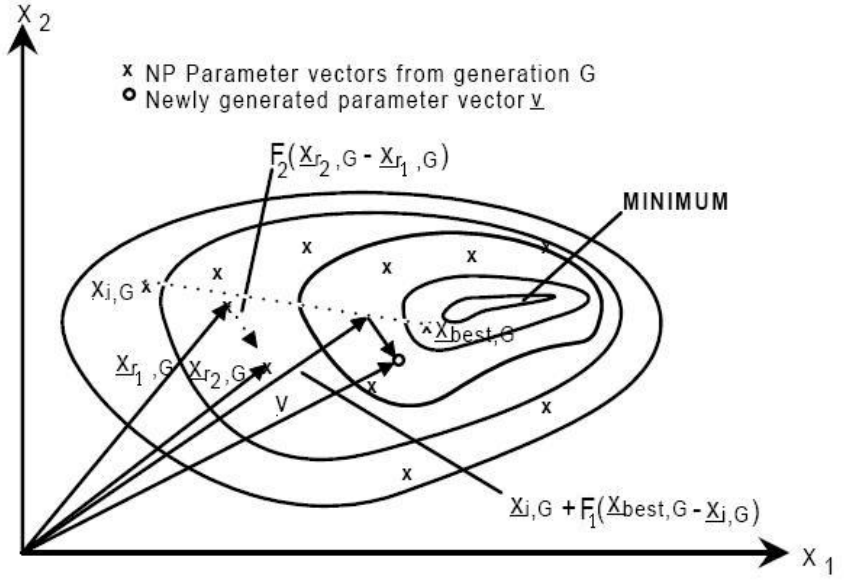


Figure 4.5, DE current-to-best representation for a two-dimensional problem [22]

MDE scheme

This method – Modified Differential Evolution [23], – works on the selection process of the vectors for the mutation. Using as basis vector a solution with good fitness, the child will inherit with high probability some of its good

properties. The modified version MDE is a kind of generalization for the *rand* and *best* method proposed by Storn and Price.

Introducing a selection pressure control variable, *PR*, the user can control the selecting process for the basis vector in accordance with the ranking of the current population made by fitness. Performing a series of independent Bernoulli trials that start from the top of the ranking, where there are the fittest solutions, the basis vector is selected according to the pressure control variable: if $U(0,1]<PR$ the current ranked individual is selected, otherwise the Bernoulli trial is repeated for the next solution in the list. The higher *PR*, the higher in the ranking is the basis vector.

The *rand* and *best* variants become a special case of MDE, respectively if $PR \leq 1/NP$ and $PR=1$. Using this method, the user can choose between the two classic variants only changing the pressure control value. High *PR* values facilitate the convergence speed, creating more fit mutated individuals, while low values, coupled with the number of weighted differences, facilitate the population diversity.

The other properties of the algorithm, like the number of weighted differences *y* and the crossover type *z* must be decided as usual.

This approach gives more flexibility to the setting of DE, but, like the others variants, it needs firstly the adjustment of *PR* and subsequently of *CR* and *NP* to achieve faster convergence speed. Moreover this tuning is, like often appear in EAs, dependant by the objective function to optimize.

DERL scheme

This technique – Differential Evolution with Random Localization [13] – is inspired by the random selection of the vectors for the mutation process, but, despite the classical formulation, it uses as basis vector the fittest solution (called tournament best solution) between the chromosomes selected for mutation. If the number of weighted difference chosen is for example one, the selection process chooses three mutually different vectors (different also from the target vector) and from them it selects the fittest one $\underline{x}_{tb,G}$ as basis vector.

$$\underline{v}_{i,G} = \underline{x}_{tb,G} + F_i \cdot \left(\underline{x}_{d_1,G} - \underline{x}_{d_2,G} \right) \tag{4.10}$$

The other two vectors, $\underline{x}_{d1,G}$ and $\underline{x}_{d2,G}$, are chosen randomly and $F_i \in [-1,-0.4] \cup [0.4,1]$. The resultant noisy vector is then passed to the crossover operator.

Using the scaling factor, even negative, with a uniform selection increases the exploration and the robustness of the algorithm but not significantly the convergence speed; moreover the choice of a working range instead a fixed value makes the setting less critical. On the other hand the employment of the tournament best as basis vector makes the algorithm faster with little

improvement in the robustness. These two effects are studied [13] before separately to proof their goodness. The resultant combination has remarkable improvement.

This random localization feature gradually transforms itself like DE, enhancing the convergence speed after a clustering around the global optimum: at the beginning the noisy vector is not necessarily local to the tournament best, because the weighted differences are large; only in the later stages, when the differences become smaller, the speed-up is significant.

Another version, greedier than the previous one, uses F_i uniformly distributed only in a positive range and the selection of the two vectors for the difference is made with the same fittest criterion: the better of the remaining two vectors will be the first chosen for the difference. In that way also the difference is directed toward a better region. This scheme could create some stagnation if a soon clustering appears: to avoid it, the exploration must be enough widespread, handling the population number *NP* or the scaling factor's interval.

These techniques seem good and non-critical: in the early stages they work like a classic DE with the necessary diversity but in the late they intensify the search.

TDE scheme

This scheme – Trigonometric Differential Evolution [24] – modifies the mutation operator in its formulation: the classic method perturbs a basis vector with a weighted difference; this one defines a search space delimited by a hypergeometric triangle formed with three vectors that represent the vertices. Using objective function information, it adjusts the perturbation toward the fittest one. The other processes in the algorithm, crossover and selection, are performed as usual.

Choosing randomly three mutually different solutions from the current population $r_1, r_2, r_3 \in \{1, 2, \ldots, NP\}$, different from the target vector as in the classic formulation, it uses as basis vector the centre point of the hypergeometric triangle formed instead only one of them. The amount of perturbation is driven by fitness information of each vector: precisely, there are three weighted differences scaled with the difference of the goodness of the vectors involved into the perturbation. The following formulation explains the process for a minimization task:

$$p_k = \frac{\left| f\left(\underline{x}_{r_k,G}\right)\right|}{\sum_{k=1}^{3}\left| f\left(\underline{x}_{r_k,G}\right)\right|} \tag{4.11}$$

where the best solutions has the lowest index p_k.

$$\underline{v}_{i,G} = \frac{\left(\underline{x}_{r_1,G} + \underline{x}_{r_2,G} + \underline{x}_{r_3,G}\right)}{3} + \left(p_2 - p_1\right)\cdot\left(\underline{x}_{r_1,G} - \underline{x}_{r_2,G}\right) + \\ + \left(p_3 - p_2\right)\cdot\left(\underline{x}_{r_2,G} - \underline{x}_{r_3,G}\right) + \left(p_1 - p_3\right)\cdot\left(\underline{x}_{r_3,G} - \underline{x}_{r_1,G}\right) \tag{4.12}$$

that could be re-write in a general form as follows:

$$\underline{v}_{i,G} = \sum_{k=1}^{3}\left\{\frac{\underline{x}_{r_k,G}}{3} + \underline{x}_{r_k,G}\cdot\left[\sum_{w=1}^{3}\left(p_w - p_k\right)\right]\right\} \tag{4.13}$$

If the scope of the optimization is the maximization, the formulation needs only the sign inversion for the scaling factors.

Thanks to the weighted terms, the noisy vector is direct toward the fittest vertex of the triangle with a greedy perturbation grade: the higher the difference in the fitnesses, the larger the amount of perturbation in a good direction. This greediness could be seen as a problem for the population diversity, since this mutation is a kind of local searching technique; for this reason the TDE operator is used only with a probability according with a new control parameter M_t. Certainly this modification helps the accuracy of the algorithm significantly, but if it is used without care it could destroy the robustness of the evolution process giving a premature stagnation. In fact, when M_t=1, the stagnation is almost unavoidable: the operator can explore only inside a predefined region, whose extension depends on the population diversity. The recommended value for this control parameter is around 0.05, rather low, but as other strategies, the tuning depends on the problem's nature. It seems that TDE introduces another control parameter M_t besides the three parameters scheduled in the original DE, *F*, *CR* and *NP*, but a look into the sensitivities between each other shows a remarkable behaviour of TDE, which appears with a considerably lower sensitivity to the variations in the control parameters. The Figure 4.4 shows the sensitivity of the two algorithms with the Rastrigin's function (see Appendix A, f22).

For this reason, after the trigonometric control parameter tuning, the algorithm setting becomes easier. The application of the trigonometric operator could be probably appropriate only in the later stages, since in the early exploration its contribution is rather low; after a clustering around the better regions of the feasible area, this operator increases the convergence speed significantly.

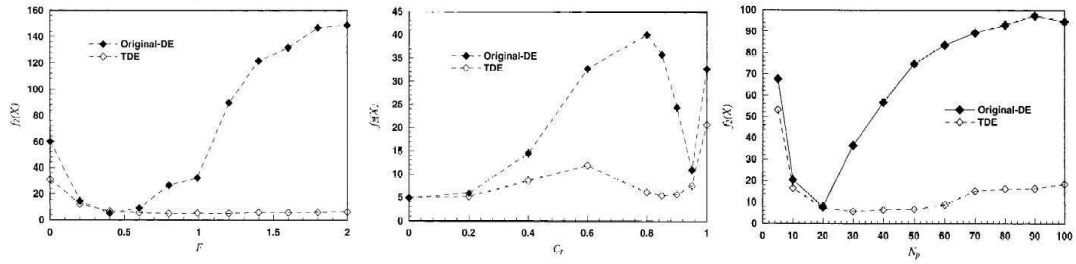


Figure 6.4, Sensitivities of the three DE parameters F, CR and NP on the Rastrigin's function for the final minimum obtained [24].

DEPCX scheme

This technique – Differential Evolution with Parent Centric crossover (X) [25-26-27] – uses a similar concept of TDE: it enhances the chance of exploring the neighbourhood more efficiently by a different noisy vector formulation; nevertheless, it does not involve fitness's information. Although the name goes back over the crossover concept, this approach works only on the mutation process; the misunderstanding arises from the first formulation of this operator, used in a particular version of GA. DEPCX differs from DE only in the mutation process. Crossover and selection are performed as usual.

The operator, unlike TDE, uses as basis vector one of the μ mutually different vectors $r_1,\ldots,r_\mu \in \{1,2,\ldots,NP\}$ selected for the mutation process, $\underline{x}_{p,G}$, $p \in \{1,2,\ldots,\mu\}$.

The mean vector is computed as

$$\underline{g} = \frac{1}{\mu}\sum_{k=1}^{\mu} \underline{x}_{r_k,G} \tag{4.14}$$

and the direction vector as

$$\underline{d}_{p,G} = \underline{x}_{p,G} - \underline{g} \tag{4.15}$$

From the remaining μ-1 chromosomes, the perpendicular distances D_i to the line of the direction vector are computed and their average $\overline{D}$ is obtained.

Then the noisy vector is found as

$$\underline{v}_{i,G} = \underline{x}_{p,G} + w_\varsigma \underline{d}_{p,G} + \sum_{k=1,k\neq p}^{\mu} w_\eta \overline{D}\underline{e}_k \tag{4.16}$$

where $\underline{e}_k$ are the orthonormal bases perpendicular to $\underline{d}_{p,G}$, $w_\varsigma = N(0,\sigma_\varsigma^2)$ and $w_\eta = N(0,\sigma_\eta^2)$. The two variances are usually taken as 0.01.

Like TDE, DEPCX is a greedy operator, but with a less forced approach. For this reason it could be applied every mutation for each target vector; the stochastic application of DEPCX, pursued also in TDE with another control parameter, could be implemented with similar probability (0.01) but with comparable results. Like classic DE, this technique is self-organized, because the amount of perturbation is function of the population diversity that affects the direction vector and the average perpendicular distance, but in the later stages it works as a local-search operator. Furthermore, when the parents selected for the mutation are located far from each other, the noisy vectors generated are well sparsely located, so at the beginning DEPCX works, unlike TDE, maintaining the necessary diversity to explore the entire domain efficiently.

In terms of solution accuracy, this algorithm is not better than other DE strategies, however, it shows faster convergence since it takes less function evaluations. A debatable problem is the necessary computational time, comparable if not greater than DE and TDE, because of the complexity of the DEPCX mutation process. To avoid this situation the stochastic application is preferable, maintaining at the same time the robustness and the convergence.

NSDE scheme

This approach – Neighbourhood Search for Differential Evolution [28] – is inspired by the exploration methods used in the Evolutionary Programming (EP). As seen before with DERL, TDE and DEPCX, the ability of an evolutionary algorithm to explore the neighbourhood space is a remarkable advantage. In this approach, the mutation operator is modified to allow a similar exploration performed in EP, where a kind of self-adapting perturbation is used. In DE this role is played by the vector differences employed in the mutation process that automatically adapt themselves.

An example of perturbation in EP is:

$$\underline{v}_{i,G} = \underline{x}_{i,G} + \underline{\eta}_i \cdot \psi \tag{4.16}$$

where ψ is the specific EP-operator and $\underline{\eta}_i$ is the auto-adaptive perturbation that follows some defined rule.

ψ could be:

- $\underline{N}(0,1)$ for classical evolutionary programming CEP, where $\underline{N}(0,1) = \left(N_1(0,1), \ldots, N_n(0,1)\right)$, that is an array of Gauss random variable with mean = 0 and variance = 1.
- $\underline{\delta}(0,1)$ for fast evolutionary programming FEP, where $\underline{\delta}(0,1) = \left(\delta_1(0,1), \ldots, \delta_n(0,1)\right)$, that is an array of Cauchy random variables with location = 0 and scale = 1.

- $\underline{L}(c)$ for Lévy evolutionary programming LEP, where $\underline{L}(c)=\left(L_1(c),\ldots,L_n(c)\right)$, that is an array of Lévy random variables with scale $0 < c < 2$.

These are the most used operators for EP in which the neighbourhood concept dominates the perturbation.

In classical DE the operator is the fixed F value.

In the NSDE approach, after a probability evaluation of a jump-length expected [28], dependant on the operator, a flexible setting of operators is performed.

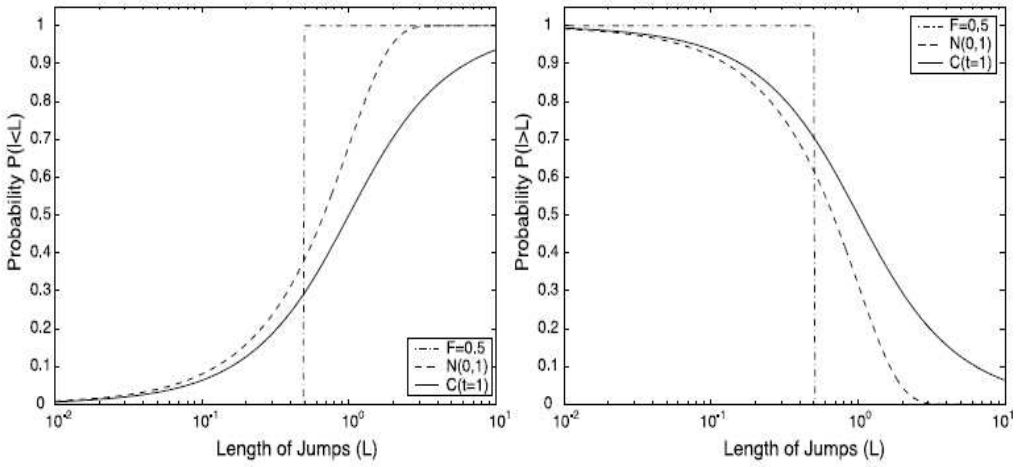


Figure 4.7, Probabilities for the lenght of jumps for three scaling factor definitions: fixed, Gauss random variable and Cauchy random variable. [28]

Since using a fixed scaling factor reduces the probability that the step length is right for the optimization problem without any tuning, to increase the ability of the algorithm two operators, $\underline{N}$ and $\underline{\delta}$, are applied with some deterministic rule; the Gaussian operator is more likely to produce small jumps, beneficial for the local search near the global optimum, whereas the Cauchy operator is more expandable and it has greater probability to produce long jumps, useful for the early exploring phase and to escape from local optima.

The following rule is proposed:

$$\underline{v}_{i,G}=\underline{x}_{r_1}+\begin{cases}\left(\underline{x}_{r_2}-\underline{x}_{r_3}\right)\cdot N(0.5,0.25) & \text{if } U(0,1]<NS\\ \left(\underline{x}_{r_2}-\underline{x}_{r_3}\right)\cdot \delta(0,1) & \text{otherwise}\end{cases} \tag{4.17}$$

Where as usual $r_1, r_2, r_3 \in \{1, 2, \ldots, NP\}$ are chosen randomly with $i \notin \{r_1, r_2, r_3\}$, NS=0.5 and the Gaussian operator is adjusted as the common range for F.
Of course the weights of the Gaussian and Cauchy operators in the perturbation could be changed modifying the *NS* value, changing the behaviour of the algorithm; in fact this value becomes another parameter to set, like the parameters for the operators.

The tests performed in [28] show the goodness of this approach versus the classical DE and FEP, with some doubt for multimodal functions with smaller number of local optima: in such cases DE outperforms NSDE, demonstrating the unhelpfulness of the NS operator, which increases only the computational time without any advantage. In all the other cases NSDE results better than classic DE, more powerful when searching in an environment without any previous knowledge.

DELN scheme

This technique – Differential Evolution with Local Neighbourhood [29] – is a variant for the mutation process that is inspired by the Particle Swarm Optimization, another evolutionary algorithm that simulates the social behaviour of the imitation of the fittest. The connection between these two types of optimization algorithms is the simplicity of the perturbation introduced in the population. Like NSDE, it increases the DE neighbourhood searching properties combining them with global exploration.

This approach mixes local and global search technique as follows:

$$\underline{v}_{i,G} = w_G \cdot \underline{g}_{i,G} + (1 - w_G) \cdot \underline{l}_{i,G} \tag{4.18}$$

where $w_G \in [0,1]$ is a scaling factor applied to the global and local mutated vectors.

These two mutated vectors are generated using the *current-to-best* approach presented in the Storn and Price variants, with the following shrewdness for the local one: after the definition of a nearness variable for each target vector $k_i \in \{1, 2, \ldots, NP\}$, a new sub-set could be defined from the main population for each target vector:

$$S_{Li} = \left\{ \underline{x}_{\langle i-k_i-1 \rangle_{NP}+1}, \underline{x}_{\langle i-k_i \rangle_{NP}+1} \ldots, \underline{x}_{\langle i+k_i-1 \rangle_{NP}+1} \right\} \ \forall i \{1, 2, \ldots, NP\} \tag{4.19}$$

From the sub-set associated with the target vector, two vectors are selected randomly for the vector difference and the fittest one in the set is chosen as the best vector for the *current-to-best* approach.

$$\underline{g}_{i,G} = \underline{x}_{i,G} + F_1 \cdot \left(\underline{x}_{best,G} - \underline{x}_{i,G} \right) + F_2 \cdot \left(\underline{x}_{r_1,G} - \underline{x}_{r_2,G} \right) \tag{4.20}$$

$$\underline{l}_{i,G} = \underline{x}_{i,G} + F_3 \cdot \left(\underline{x}_{nbest,G} - \underline{x}_{i,G} \right) + F_4 \cdot \left(\underline{x}_{r_3,G} - \underline{x}_{r_4,G} \right) \tag{4.21}$$

where *best* indicates the best vector in the entire population, $r_1, r_2 \in \{1, 2, \ldots, NP\}$, *nbest* is the neighbourhood best solution inside the set S_{Li} and r_3, r_4 are chosen randomly from the sub-set interval. The fourth scaling factors are usually mutually different.

The weight factor varies linearly (increasing) with time, following eq. (4.22):

$$w_G = w_{\min} + \left(w_{\max} - w_{\min} \right) \cdot \left(\frac{G-1}{G_{\max} - 1} \right) \quad \forall G \in \{1, 2, \ldots, G_{\max}\} \tag{4.22}$$

After a necessary definition of the minimum and maximum value (advised values are respectively 0.4 and 0.8) to achieve the balance between local and global exploration, this time-varying approach gives emphasis at the local search in the early stages and only with time it moves toward the global search. This neighbourhood concept does not lie in a space conceptualization but only in a population point view, since a neighbour is given only by the solution's index: for this reason, the "local" wording assumes a different meaning, with less implication about the searching area: so, emphasizing at the beginning the local search doesn't mean reverse the exploration scheme in the domain.

This DELN scheme should improve the performance compared with classic DE, but it seems just a sophistication in the choice of the vectors.

4.2.2 Crossover options

The most common crossover operator for DE is the binomial crossover, explained in Section 4.1. In this section the other crossover option proposed in literature is presented.

Exponential crossover

This crossover scheme was the first proposed in [21] by Storn and Price; it takes idea form the GA crossover process in which the mixing of the genes is made defining one or multiple cutting points. As explained before, the crossover role is to maintain diversity inside the population, mixing the properties of the created noisy vector with the target vector. This process mixes the chromosomes cutting in the target vector in two points, inserting the noisy vector's genome, like the two-point crossover for GA.

The trial vector is composed by the following rule:

$$u_{ji,G} = \begin{cases} v_{ji,G} & for\ j = \langle a-1 \rangle_n + 1, \langle a \rangle_n + 1, \ldots, \langle a+L-2 \rangle_n + 1 \\ x_{ji,G} & otherwise \end{cases} \tag{4.23}$$

$$\forall j \in \{1, 2, \ldots, n\}$$

where $a \in [1, n]$ and $L \in [1, n]$ are randomly chosen integers and $\langle \ \rangle_n$ is the *n*-modulo; it permits the circularity of the process. The first index *a* defines the first cut point, whereas *L* defines the length of the replacement. In this process, the probability that *L* variables in the trial vector come from the noisy vector is

$$P(L = h) = CR^h \tag{4.24}$$

This is not a probability distribution but just a relationship where the operator can modifies the result changing *CR* or *h*, according with the power law: the probability of mutating *h* components increases with the parameter *CR* and decreases with the value of *h*. Anyway, to have a good effect with the exponential crossover, the *CR* value must be high, 0.8÷0.9.

In the case *CR* = 1 then all the parameters of the trial vector come from the noisy vector.

The weakness of this crossover method is the circularity of this approach, in fact, the exponential crossover modifies only consecutive genes. Figure 4.6 shows the process.

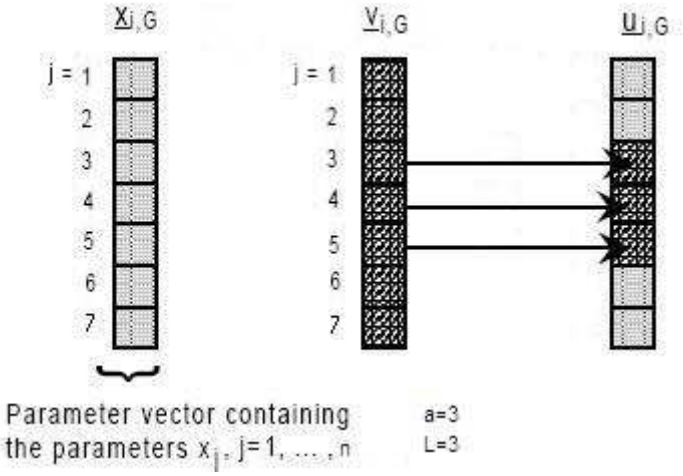


Figure 4.8, Scheme of the exponential crossover for DE [21]

4.2.3 Further sophistications

This section is dedicate to the different processes planning adopted in the algorithm's implementation; since the modifications on the operators' nature is not the only way to increase the ability of DE to explore the search space and to increase the convergence speed, this further modifications in the algorithm behaviour play an important role in the predominance of DE in comparison with other optimization algorithms.

DELB scheme

This scheme – Differential Evolution with random Localization around the Best vector [13] – adopts the further exploration by localization after the definition of the trial vector. Respect to DERL, sometimes it is preferable adopting the localization concept after the calculation of the trial point, to explore the region between it and the best current solution. This scheme is not a modification of the mutation process, rather a subsequent mutation around the trial vector and the best solution. It starts like the classic DE [21]: defined the trial vector $\underline{u}_{i,G}$, if its fitness is better than the target vector's fitness but worse than the best current vector's fitness, two new points are found with some probability ($U(0,1]<w$), using the following general rules for reflection and contraction:

$$\begin{aligned} r_{ji,G} &= \alpha_j u_{ji,G} + \left(1-\alpha_j\right)\cdot x_{jbest,G} \\ c_{ji,G} &= \alpha_j x_{jbest,G} + \left(1-\alpha_j\right)\cdot u_{ji,G} \\ &\forall j \in \{1,2,\ldots,n\} \end{aligned} \tag{4.25}$$

where $\alpha_j = U(-1,1)$, one for each *j*th variable.

A first attempt is made by the reflected vector generated: if its fitness is worse than the original trial's one, a second attempt is made with the contracted vector; if the second attempt fails again, the trial vector replace the target vector instead one of the previous twos. The user-defined *w* parameter controls the frequency of the local exploration around the trial and the best vectors. This scheme increases the number of function computation per generation, but evolutionary speaking it reduces the number of generation necessary to achieve the global optimum; furthermore, this scheme increases the robustness in view of the correct choosing of the control number *w*: the recommended value is 0.1.

In this practice the convergence speed and the robustness are increased in comparison with a classic DE, but the introduction of another variable to be adjusted attentively makes this technique really sensitive to the control

parameters. Without a correct tuning, the convergence and the robustness could be call into question.

DEPC scheme

This approach – Differential Evolution with Preferential Crossover – is proposed in [26] and it works managing the crossover operator and the selection process. It uses two populations, both driven toward the global optima. The idea of a second population comes from the strict rule of the selection process: in DE the trial vector is discarded if the target vector shows better fitness, but this does not mean that the trial vector is completely inefficient: it could be better than other individuals inside the population and it may even lie in a promising region of the search space.

In DEPC, the first crossover procedure is applied using two populations, S_1 and S_2 with the classic method. The set S_1 is the main population, from which the target vectors come, and the second set S_2 is a parallel population within the discarded trial vectors are stored. Both the sets are initialized uniformly inside the domain. The first trial vector is defined from a crossover between a uniform random point $\underline{a}_i \in S_2$ and the target vector, for example using binomial crossover:

$$u_{ji,G} = \begin{cases} a_{ji,G} & if\ U(0,1] \le CR\ \ or\ \ j = irand(NP) \\ x_{ji,G} & if\ U(0,1] > CR\ \ and\ \ j \ne irand(NP) \end{cases} \qquad (4.26)$$

$$\forall j \in \{1, 2, \ldots, n\}$$

After the definition of the first trial vector, the classic selection rule takes place; otherwise another attempt is made with a second trial vector $\underline{u}_{i,G}$': the second trial is produced by the crossover between a noisy vector, deriving from a mutation rule that involves solutions from S_1, and the target vector. Then, another selection process between the target and the second trial vector is made. If also this time the trial results worse, it is not abandoned altogether and it competes with the corresponding target vector in S_2.

This approach enhances the probability to generate feasible points inside the domain, decreasing the function evaluations, the cputime and slightly increasing the success rate in comparison with classic DE.

For this reason is recommended in the high-constrained problems where the feasible area is quite difficult to individuate by the algorithm.

ODE algorithm

This approach – Opposition-based Differential Evolution [30], [31], [32] – employs the opposition-based optimization OBO [19] concept for the population initialization and generation jumping.

DE, like other evolutionary algorithms, starts with a set of candidate solutions and it tries to improve them toward the optimum. Without any previous information, the initialization is made random. Since the speed of an algorithm is given by the distances between the initial candidate solutions and the true optimum, the ODE approach permits to enhance the probability to start from better solutions checking also the opposite of the initial population. According to the probability theory, 50% of the time a guess is farther from a solution than its opposite: for this reason, after the initial guesses also the opposite solutions in the search space are checked and used as initial solution. This veracity is proved in [32] and applied to DE, not only for the initial population of the algorithm.

To define the opposite let x be a real number inside a defined interval (without the interval the opposite concept cannot be take into account) $x \in [a,b]$, then its opposite is

$$\breve{x} = a + b - x \tag{4.27}$$

This definition can be extended to higher dimension:

$$\underline{x} = (x_1, x_2, \ldots, x_n) \tag{4.28}$$

is a point, defined by a vector composed by n-real numbers, each of them contained in a specific interval, defining the domain D:

$$x_j \in \left[x_j^L, x_j^U \right] \quad \forall j \in \{1, 2, \ldots, n\} \tag{4.29}$$

Then the opposite point is defined by a vector composed by the opposite real-numbers:

$$\underline{\breve{x}} = (\breve{x}_1, \breve{x}_2, \ldots, \breve{x}_n) \tag{4.30}$$

$$\breve{x}_j = x_j^L + x_j^U - x_j \quad \forall j \in \{1, 2, \ldots, n\} \tag{4.31}$$

In ODE, after a random initialization (G=1) of the population S, the opposite population set OS is calculated. Then the evaluation of the two population sets is computed and the fittest individuals are selected as initial population for the algorithm (the so called *elitist selection* for GA). This process increases the probability to start close to the solution, decreasing the total computational time, even if for the first population are necessary $2 \cdot NP$ fitness evaluations. Another strategy introduced in ODE is the opposition-based generation jumping:

according with a jumping probability value J_r, after the classic operations' flow – mutation, crossover and selection – if $U(0,1)<J_r$ the opposite of the current population is computed and, as in the initial population selection, the fittest solutions are selected from the union of the two sets. A little shrewdness is introduced, making the process dynamic: the opposite is not computed using the initial upper and lower values for each variable, but rather using the maximum and minimum values of each variable at the current population's state:

$$\breve{x}_{ij,G} = \min_{j,G} + \max_{j,G} - x_{ij,G} \qquad (4.32)$$
$$\forall i \in \{1,2,\ldots,NP\}, \forall j \in \{1,2,\ldots,n\}$$

then without losing the knowledge acquired by the algorithm till this moment; this approach permits to decrease the searching space during the opposition-based generation jumping. A similar device will be treated in Section 4.4.

The jumping rate J_r could be fixed (defined by the user) or time-varying. In [30] the better algorithm's behaviour is given by a jumping rate decreasing by function calls; in particular J_r follows the rule

$$J_r = \left(J_{r_{\max}} - J_{r_{\min}}\right) \cdot \left(1 - \frac{nfc}{MAX_{nfc}}\right) \qquad (4.33)$$

where *nfc* is the number of function calls and MAX_{nfc} is the maximum value allowed for *nfc*.

The jumping rate parameter range recommended is 0.1÷0.4; higher values could destroy the evolution, provoking early stagnation due to the shrinkage of the search space; also the time-varying approach will be explained further in Section 4.4.

ODE with fixed or time-varying jumping rate outperforms the classic DE in many situations [30], demonstrating the strength of the OBO technique [32] successful combined with the DE evolutionary properties. The improvement is reflected principally toward a less number of function calls, to the detriment of the success rate: this means the robustness is a little penalized, but the reduction in *nfc* is remarkable.

MDE algorithm

This approach – Modified Differential Evolution [33], different from MDE of Section 4.2.1 – arises from the necessity to reduce the computational time of DE. Although DE is efficient, effective and robust, it has a lack in the convergence speed for high dimensional problems. This weakness is made less heavy by MDE algorithm: it uses a slight strategy on the selection ruling for the next generation: instead performing the selection and the possible substitution

after mutation and crossover for the whole current population, thus creating a temporary population of trial vectors, the target vector is immediately substituted if the trial vector appears fitter than it. Using this device, the current population evolves dynamically and not with discrete generation step, because the benefitting is dynamically updated. This algorithm does not upset the DE classic method, rather it modifies the moment of the updating process, making the algorithm faster and at the same time robust as the classical formulation. This approach can be advantageous in real-world problems, where the evaluation of a candidate solution is a computationally expensive operation. A further sophistication of this method is proposed in [34], where the dynamic-substitution is combined with ODE and the mutation operator is taken from the DERL scheme. Of course mixing correctly strategies with different effects increases the ability of the algorithm, especially in high dimensional cases.

NSDE algorithm
This algorithm – Non-linear Simplex Differential Evolution [35], different from the previous described in Section 4.2.1 – uses an approach similar to the OBO applied at the initial population: the aim is to decrease the distance between the initial population and the true optimum of the problem in order to diminishing the optimization time. It uses a non-linear simplex mutation to the initial random population so as to create another set of candidate solutions: the fittest individuals form the random and the modified sets belong to the initial population. NSDE, in comparison with ODE, acts only on the initial population setting, without any further modification in the classic algorithm: its role is just to increase the probability of obtaining the optimum in fewer function calls affecting the starting point.

This method is simple, it uses the simplex formulas [2] of reflection, contraction, expansion and reduction and it does not touch the algorithm behaviour, leaving the robustness and the efficiency of DE intact. In the definition of the initial population it takes more computational time but it provides a better initial condition, allowing a successive efficient function call saving; the algorithm shows slightly better behaviour than ODE in terms of *nfc* and cpu in most cases, even if the sophistication adopted seems insubstantial in comparison with ODE. Extra-information about the search space to the initial population are beneficial, enhancing the convergence speed without compromising the robustness.

The main structure of DE could be mixed or hybridized with other methods: for example, applying to a population based search algorithm a further classic optimization method based on gradient evaluation as speed up, the efficiency of the search method could be increased significantly without loosing the robustness of the evolutionary procedure. Some successful examples are

presented in [35], where DE is hybridized with a quasi-Newton method, and [36], where Particle Swarm Optimization is integrated with DE.

4.3 Constrained optimization

The handling constraints problem is typical for real-optimization problems: usually the difficult in the real world is to find the solution that satisfies all the constraints accordingly with a high performance.

If the handling constraints has direct or easy reference to the domain of the variables, it is sufficient repair the solutions created outside the domain. In that way, if during the evolution process some trial vector is created by the mutation outside the domain, some repair rule is adopted.

The rules most used are:

- *repairing to the bounds*

$$v_{ij,G} = \begin{cases} x_j^L + \delta \cdot U(0,1) & if\ v_{ij,G} < x_j^L \\ x_j^U - \delta \cdot U(0,1) & if\ v_{ij,G} > x_j^U \end{cases} \tag{4.34}$$

  where δ is a small number, around 0÷0.1. This feature is used when it's easy going outside the domain because the optimum is collocated close to the boundary; of course this rule doesn't allow diversity in the regeneration and the amount of δ could influence the searching technique; if $\delta = 0$, the trial vector is placed exactly on the border;
- *repairing randomly inside the domain*, that uses the same rule of the random initialization. This feature doesn't allow to the algorithm to remember about the direction of the evolution, restoring the initial condition. If the optimum is close to the boundary the algorithm could have fewer chances to find it.
- *repairing bouncing back* the excess outside the domain produced by the mutation process: this approach is recommended especially for the situation in which the optimum could be very close to the border of the domain or even on the border.

$$v_{ij,G} = \begin{cases} x_j^L + \left| v_{ij,G} - x_j^L \right| & if\ v_{ij,G} < x_j^L \\ x_j^U - \left| v_{ij,G} - x_j^U \right| & if\ v_{ij,G} > x_j^U \end{cases} \tag{4.35}$$

If the handling constraints cannot bring back to the domain definition, as in many real-problems, but it depends on some resultant property of the system, several other methods were proposed, especially for GA but applicable to DE.

Michalewicz [37] grouped these methods in four categories:

1. methods based on preserving feasibility of solution
2. methods based on penalty functions
3. methods which make a clear distinction between feasible and infeasible
4. hybrid methods

Methods based on preserving feasibility of solution

The first category is based on specialized operators which transform infeasible solutions into feasible ones; this method unfortunately accepts linear constraints only and needs to start form a feasible initial population. Into this category lie also the repair rules, since these operators work in a similar manner, determining the current domain $\Omega_l \subset \Omega \subset D$ that is a function of the linear constraints and repairing the solution. Some different approach could be applied for nonlinear constraints or optimum close to the boundary, but in this case is necessary implement specific operation strategies related to the problem's nature.

Methods based on penalty functions

The second category is based on a penalization imposed to any infeasible solution, reflecting the infeasibility directly into the fitness function, the only one submitted to selection in the algorithm; this practice is similar to the multi-objective optimization, described in section 4.6, in which all the properties to optimize are collected in a unique overall function. As for MO problems, the main difference in this method depends on how the penalty (or overall function) is designed.
The common rule is:

$$f_k\left(\underline{x}_{i,G}\right) = \begin{cases} f_k\left(\underline{x}_{i,G}\right) & \text{if } \underline{x}_{i,G} \in \Omega \\ f_k\left(\underline{x}_{i,G}\right) + penalty\left(\underline{x}_{i,G}\right) & otherwise \end{cases} \tag{4.36}$$

Usually the penalty is a function based on the distance of the solution from the feasible region, so it is the result of the combination of all the constraints. For this reason there are C function $f_m\left(\underline{x}_{i,G}\right) \quad \forall m \in \{1,2,\ldots,C\}$ used to build the penalty function.

There are several ways to design the penalty function and of course the difficulty is to define the appropriate feature of this additional function. The strategies proposed in literature are:

- Static penalty, where usually the penalty is the sum, the averaged sum or the weighted sum of all the constraints. For this method the design is high dependant on the nature of the constraint: if it is just related to some input variable this approach could be efficient if correctly calibrated, otherwise,

if the constraint is a property of the system the value must be normalized if possible and attentively weighted. The number of additional parameters could be too high.
- Dynamic penalty, where the number of parameters involved is low and the penalty amount is time dependant; this approach could return at the end an infeasible solution.
- Annealing penalty, that is based on dynamic penalty method with annealing behaviour.
- Adaptive penalty, where the amount of the penalty depends on the population condition referred to the feasible region: if many points lie out of the feasible region the penalty is high, otherwise the penalty is relaxed.
- Death penalty, that simply rejects the infeasible solutions; this method could be impracticable in high-constrained problem. This approach could be seen in the third category.
- Segregated penalty, that implements two different penalized fitness function with different weights and picking up alternatively from the two resulting sets.

Methods which make a clear distinction between feasible and infeasible

The third category includes few methods, since these practices are not often applied. An approach is called behavioural memory method and it considers only one constraint at the time. This method seems poor in satisfaction, since a correct sequence of non-interconnected constraints must be defined: if the satisfaction of one constraint works again another previous one, the solution is rejected and the algorithm could converge slowly.

A most interesting method is the method of superiority of feasible points, which is based on a classical penalty approach (in fact it could be seen in the previous category) with an additional function that influences the infeasible solutions using a heuristic rule. Thereby any feasible solution is better than any infeasible, since sometimes in penalty methods an infeasible could be better than a feasible one if the penalty is low. This method seems convenient but for some problems it may have some difficulties in locating a feasible solution.

The last commonly method used of this category repairs the infeasible solutions: it works at the beginning with specialized operators that create feasible solutions only for the linear constraints. This set of solutions $S_l \in \Omega_l$. The successive step is to modifying the nonlinear equality constraints into nonlinear inequality constraints simply subtracting the precision of the system ε. Then another set S of fully feasible solutions is created and used as reference to which the set S_l is directed: taking a solution from S_l, if it is fully feasible it could be moved to the set S or left in S_l, otherwise a fully feasible solution from S is taken as reference and some points between them are evaluated till the finding of another fully feasible point. For this purpose the bisection method is

recommended. After that, the new feasible point could replace the individual in S_l or S according with some probability. This feature depends on the dimension of the fully feasible region and how the algorithm is implemented.

Hybrid methods

Into the fourth category many types and evolutions of constraints handling methods lie: it is easy hybridizing evolutionary computation with deterministic, procedures or gradient approaches. Also combining the previously mentioned methods is equivalent to implement a hybrid method.

An important example [26] of hybrid method is the approach in which violation and objective function are evaluated separately: this is a special case of the method of superiority of feasible solutions that doesn't involve a penalization in the objective function: it just optimizes constraints and objective in lexicographic order in which constraint violations precede the objective function. In this approach the constraint handling could be strictly constant or relaxed during evolution like other strategies.

A practicable way to handling the constraints is to adopt the multi-objective optimization in which the constraints are treated as objective function. Of course this method could be more difficult since the nature of MO problems.

Concluding, is difficult finding the correct constraint-handling method a priori since the efficiency and convergence speed of the constrained evolutionary algorithm is heavily dependant on the problem's nature. Without any previous information about the extension of the feasible area referred to the domain, the choice of the method is difficult. The complexity of the constrained problem depends on the complexity of the objective function combined with the sparseness or shape of the feasible region like: the objective function has many local optima, the global optima is located close to the boundary, the slope of the constraints is high close to the border and the global optima. In these situations classical constraint-handling with penalty function (the easiest method to implement) could fail, making the choice really hard; for this reason the previous estimation of a constraint-handling method is still now an open question.

4.4 Control parameters' setting

The control parameters' setting of classic DE, proposed by Storn and Price, seems easy because of the small number of controls on the algorithm: *NP*, *F* and *CR*.

Despite the first recommendations given by creators, finding a correct setting is one of the most difficult tasks for the user. The robustness of D is proved in a wide range of control settings, but the convergence speed in some problems is an open question. The ability in floating-point encoding of DE over continuous space is a matter of fact, but like all the EAs, this algorithm is sensitive to the control setting. Besides, even the robustness of the algorithm could be diminished if the parameters' setting is not taken into account, causing a premature convergence due to stagnation [38].

The correct setting is strongly dependant on the problem nature, since the perturbation introduced in the population is function of the control parameters.

Just for example, Figures 4.8, 4.9 and 4.10 show the effect of the three parameters, evaluated separately, on the function evaluations (fe), the success rate (sr) and the cputime (cpu) used by DE on the four dimensional Cosine Mixture Problem, Breiman and Cutler, 1993 (Figure 4.7 shows the two dimensional problem):

$$\begin{aligned} &\min_{\underline{x}} f(\underline{x}) = \sum_{i=1}^{n} x_i^2 - 0.1 \sum_{i=1}^{n} \cos(5\pi x_i) \\ &-1 \le x_i \le 1,\ i \in \{1, 2, \ldots, n\} \\ &for \quad n = 4 \quad f(\underline{x}^*) = -0.4 \quad at\ origin \end{aligned} \tag{4.37}$$

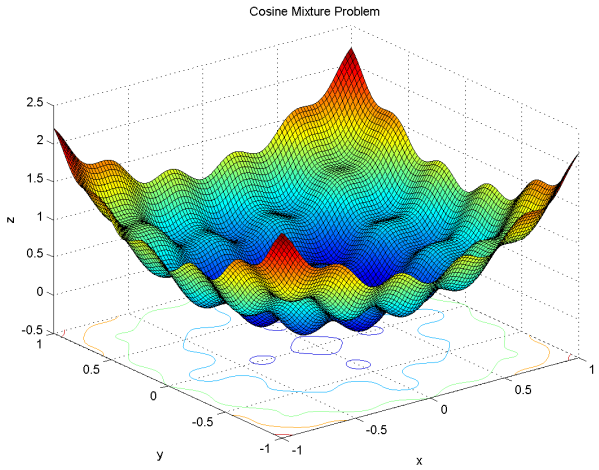


Figure 4.9, Cosine mixture problem for a two dimensional problem.

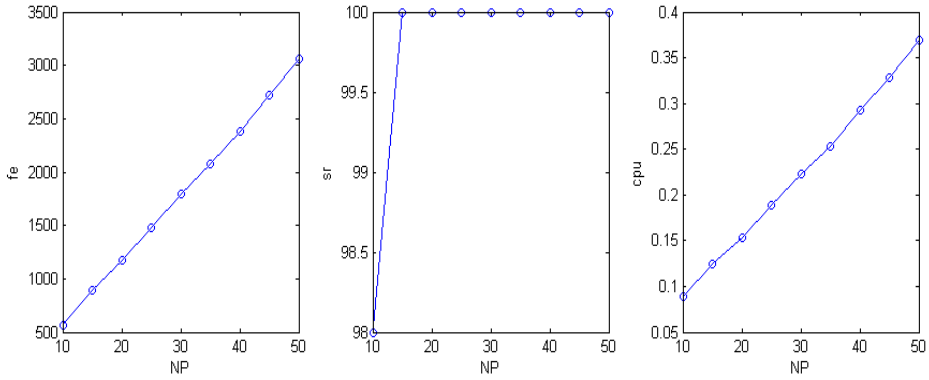


Figure 4.10, Population size effects on the three measures: fucntion evaluations, success rate and cputime.

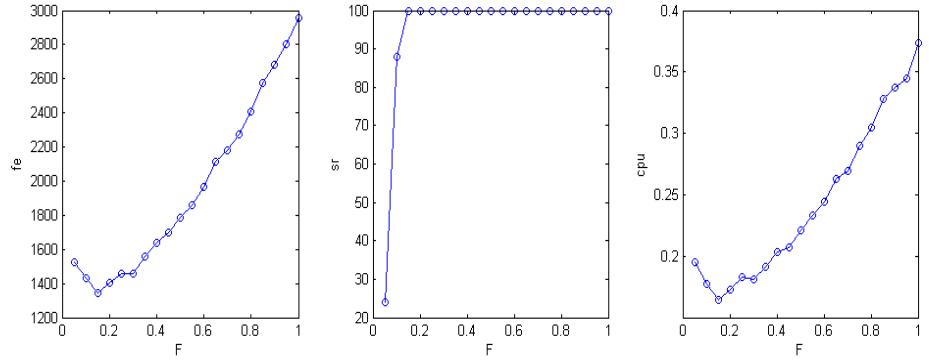


Figure 4.11, Scaling factor effects on the three measures: fucntion evaluations, success rate and cputime.

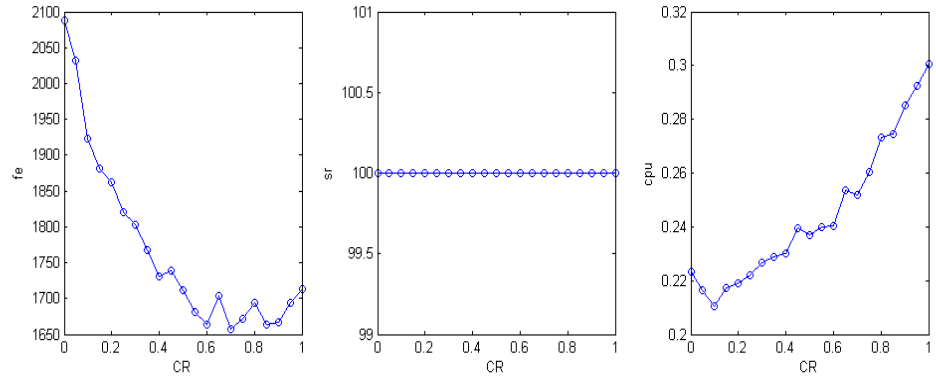


Figure 4.12, Crossover rate effects on the three measures: fucntion evaluations, success rate and cputime.

The results are related to the four dimensional problem of the Cosine Mixture Problem (see also Appendix A, f15), and DE is implemented in classical configuration [21] with F=0.5, CR=0.5, NP=30. The effects are averaged over 50 runs for each setting, in order to obtain significant statistical values respect to the randomness, and they are showed modifying just one parameter by time, holding the other two. A run is considered terminated with success if difference between the true optimum and the optimum find by DE is less than 10^{-4} in 500 generations; the algorithm halts when the fitness-difference between the best and the worst individual in the population is less than 10^{-4}.

It is clear the setting, even for the classical formulation, could improve the convergence speed, but to know the correct combination between these three factors a previous tuning of the algorithm is necessary; this practice is of course time-consuming.

NP effect

The rule of thumbs about this parameter says the recommended value is $NP=10 \cdot n$. This value is related to the nature of the perturbation process, in which the vectors selected must be randomly chosen and mutually different. To allow sufficient diversity to the next generation is necessary a selecting pool enough wide. This parameter affects the number of function evaluations of DE: if the population dimension is too high, the algorithm could waste time without any benefit. In the case previously presented, with n=4, using NP=40 is the wrong choice because the fastest convergence is found in correspondence with NP=15, significantly far from the recommended value (see figure 4.8). In this situation, after the value NP=15, the success rate becomes 100% and the higher NP, the higher fe and consequently cpu.

F effect

There is not a unique recommendation about the setting of the scaling factor; this value is heavily dependant on the problem's nature, more than the population size, since it drives the weight of the vector differences used in the perturbation process. Farther, the setting depends on the perturbation operator implemented. In our situation all the values tested over F=0.15 of the scaling factor allow a success rate of 100%; for F=0.05 the success rate is slightly more than 20%. This behaviour, caused by a small perturbation, doesn't allow the necessary exploration abilities to the algorithm, slowing down the convergence speed (the maximum number of generation allowed in this test is MAXGEN=500).

The first operating interval recommended for F was (0,2], but after some studies this interval was reduced to (0,1].

The role of F is heavily significant in the convergence speed and the success rate, for this reason, this parameter is modified in some DE variant:

- $F_i \in [-1,-0.4] \cup [0.4,1]$, proposed in DERL [13], within the uniformly distributed choice is made for each i^{th} solution.
- $F_{dither\_i/G} = F_l + U(0,1] \cdot (F_h - F_l)$, called Dither approach, mentioned in [39], in which the scaling factor could be generated for each i^{th} chromosome or G^{th} generation; in this configurations the user has to define upper and lower allowable values for F. This is quite similar to the previous approach.
- $F_{jitter\_i,j} = F \cdot (1 + \delta \cdot (U[0,1) - 0.5))$, called Jitter approach, mentioned in [39], in which the scaling factor is generated for each dimension of the problem. It seems very important using a small value of δ=0.001, in order to explore the squared neighbour around the noisy vector generated with fixed F.
- $F \sim N(0.5, 0.25)$ or $F \sim \delta(0,1)$ according with some probabilistic rule, proposed in NSDE [28]. This shrewdness could be pejorative because of the presence of other control parameters.

Randomize the scaling factor according with some distribution seems sometimes useless or with less advantages, but it appears particularly practical with noisy functions, despite the necessity to define other control parameters and values for the randomization.

CR effect

The effect of this parameter, like the others, depends on the problem nature. For the Cosine Mixture Problem, *CR* does not have effect on the success rate but only in fe and cpu. After the threshold of 0.5, the number of function evaluations does not increase but shows an assessment between 1650 and 1700 fe; only the cputime increases, since the computation of the noisy vector's dimensional element in the implemented program is made only if the crossover process is successful. The value of 0.5 is a right compromise for the two properties.

In this problem the function shape is sufficiently easy even if the cosine component modifies the parabolic trend adding local minima. Low values of *CR* decrease the convergence speed, because the mutation process induced by the perturbation of the weighted difference is highly beneficial: reducing the crossover rate means slowing the evolution process. Anyway this behaviour could not be found in other problems, especially for noisy functions with a high number of local optima.

About the setting of *CR*, Zaharie [40] handles the problem after some theoretical evaluations for the two type of crossover: binomial and exponential. These results are quite important in the user choice of crossover type and rate, because of the different rules adopted in the two variants.

The probabilities that a component is mutated are respectively:

- Binomial $p_m = CR(1-1/n)+1/n$ (4.38)
- Exponential $p_m = \frac{1-CR^n}{n(1-CR)}$ (4.39)

where n is the dimension of the problem.

Since the user wants to control the number of the mutated components, it could use as indicator the expected value $E(L)$ of mutated L components, simply $E(L) = n \cdot p_m$:

- Binomial $E(L) = CR(n-1)+1$ (4.40)
- Exponential $E(L) = \frac{1-CR^n}{1-CR}$ (4.41)

As said before, binomial crossover is a discontinuous operator while exponential is continuous; so, for the first variant, L is not the length of the chromosome replaced but the number of chromosomes inherited from the noisy vector.

The trend of the two p_m is presented in figure 4.11 for three dimensionalities, respectively n=5, n=10, n=30.

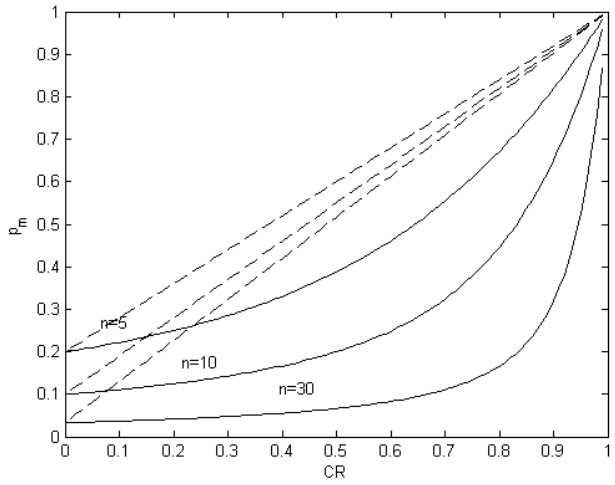


Figure 4.13, Mutation probabilities for binomial (dashed line) and exponential (solid line) crossover for three dimensionality. [40]

For low dimensionalities, the difference between binomial (dashed line) and exponential (solid line) is not remarkable, but for high dimensionalities, like n=30, the probability of mutation is significantly sensitive to the crossover rate

imposed for the exponential one. In fact, binomial crossover follows a linear trend, whereas exponential has the typical shape of the power law. It is deducible from the figure 4.11 that both exponential and binomial start from $p_m=1/n$ if $CR=0$ and finish with $p_m=1$ for $CR=1$: at least one component of the trial vector will be taken from the noisy in the first CR setting and the trial vector is in fact the noisy one for the second CR value.

Concluding about the crossover variants, it could be said that the exponential crossover, to have any significant effect with high dimensionality, needs an accurate tuning, whereas the binomial one is less sensitive to small changes and allows an easy setting. In particular, the exponential crossover for problems with high dimensions becomes significant only with values $CR \in (0.9,1]$, otherwise its effects are negligible, slowing the convergence speed. For this reasons the binomial crossover is the most used variant in DE.

Coupled effects

From the previous examples, the effects of a bad setting in terms of premature stagnation are not clear, because of the simple shape of the function; in fact, the local minima are not difficult to avoid (the weight of the cosine sum is only 0.1; increasing this value the paraboloid becomes more distorted). Only a really small value of the scaling factor causes a premature stagnation. The stagnation arises when the population lost completely its diversity and it remains unchanged by the perturbation. For this reason, to avoid premature convergence, it seems reasonable keeping a sufficient level of diversity in the population.

Zaharie in [41] proposes an important theoretical result about the coupled effect of the three parameters accounted together in a unique formulation. Unfortunately this result is related only at the classical formulation [21], with some simplification in the crossover and it cannot be applied with others DE strategies without any further theoretical evaluation.

Zaharie uses as measure of the diversity the statistical variances computed for each component over the entire population and find an interesting relationship between the control parameters and the population variance evolution:

$$E\left(Var\left(u\right)\right)=\left(2\cdot F^2\cdot CR-\frac{2\cdot CR}{NP}+\frac{CR^2}{NP}+1\right)\cdot Var\left(x\right) \qquad (4.42)$$

where $E(Var(u))$ is the expected variance of the trial vectors related to the variance of the current population. When the factor inside the brackets is greater than 1, the variance of the trial vectors should be greater than the current population variance, enhancing the exploration. Otherwise, the algorithm reduces its exploration abilities in order to find solutions close to the current population.

This result does not take in consideration the selection process, because it depends on the objective function's values; since selection usually decreases the variance, to prevent a premature diminishing of the diversity inside the population and a consequent premature stagnation, the value inside the brackets should be slightly greater than one. Nevertheless, these considerations are valid for a really wide range of objective functions, but the exclusion of the selection process leaves a significant lack of knowledge for a complete understanding.

The premature stagnation, found in the Cosine Mixture Problem, due to a small value of the scaling factor, with NP=30 and CR=0.5, could be measured by a unique control parameter, called k:

Table 4.1, k-parameter [41] and success rates for three scaling factor's setting on the optimization of the Cosine Mixture Problem (Appendix A)

F	k	Sr
0.05	0.9775	24
0.10	0.9850	88
0.15	0.9975	100

We find a success rate of 100% with a theoretical value of k smaller than 1; probably this discrepancy is due to the quite simple shape of the function and to the simplifications made in the theoretical description of the crossover operator (in the theoretical evaluation with CR=0 the p_m is 0 instead 1/n for our implementation.).

In Figures 4.12 and 4.13 are showed the contour plot for the k-parameter and the success rate of out test for $F \in (0, 0.5], CR \in [0, 0.5]$ and NP=30.

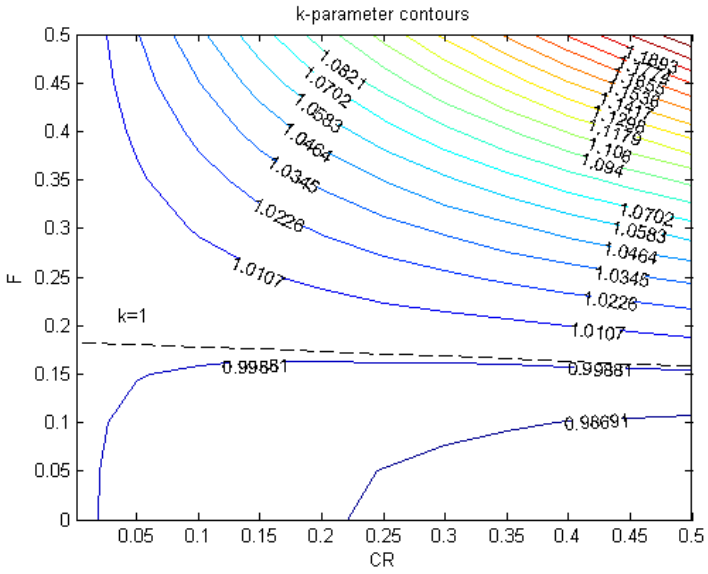


Figure 4.14, Contour plot for the k-parameter [41]. k>1 over *F*=0.2.

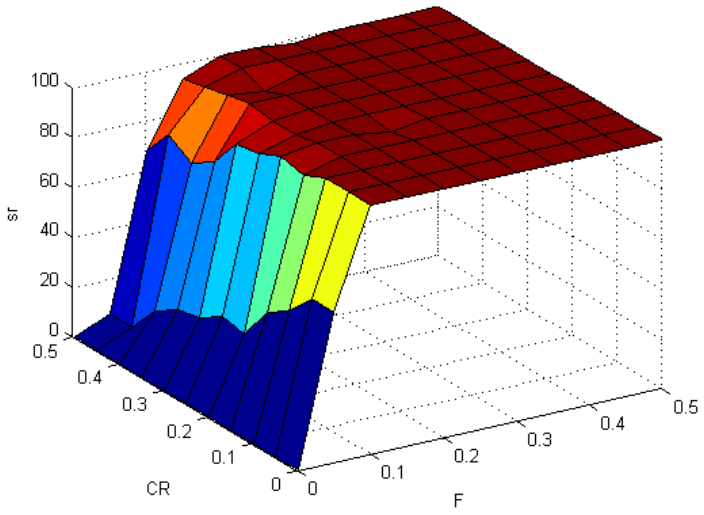


Figure 4.15, Success rate for the minimum seeking on the Cosine Mixture Problem with respect to *F* and *CR*. Sr=100% over *F*=0.2.

4.5 Adaptive and Self-Adaptive approaches for control setting

Despite the previous section describes the influence of the control parameters in the evolutionary process performed by DE, even with some interesting

theoretical results that help significantly the setting, it is clear that all the recommendation made before are limited to the DE scheme implemented and to the problem nature. Of course the basic concepts explained are valid in a general manner, but any problem needs the experience of the user to find the correct way to set the algorithm, even with all the considerations previously presented. Further, in the case the objective function is implicit rather than explicit, the tuning becomes quite difficult.

To avoid all these inconveniences and to achieve optimal convergence, these parameters need to be alterable during the evolutionary process: the tuning in that way is made directly inside the evolution.

Unfortunately all the methods proposed adjust only *F* and *CR*, since the population size *NP* is quite difficult to adapt: a fixed value, defined by the user, is always used.

The change of these control parameters can be categorized as follow:

1. deterministic parameter control
2. adaptive parameter control
3. self-adaptive parameter control

4.5.1 Deterministic parameters' control

In this setting approach one or more parameters are altered by some deterministic rule. This rule is defied by the user, giving more flexibility to the evolutionary process; an example, referred to the scaling factor, is:

$$F = \left(F_{\max} - F_{\min}\right) \cdot \left(1 - \frac{nfc}{MAX_{nfc}}\right) \tag{4.43}$$

In this case maximum and minimum values for *F* must be chosen and the maximum number of function calls must be known. This deterministic rule could be implemented in a discrete manner, using the generation number and the number of maximum generation allowable. This approach enhances the exploration in the early stages and moves toward the local search in the latter.

It could be applied with all the parameters and with all the variants of DE proposed: in fact it is the recommended practice to handle the jumping rate in the ODE algorithm [30].

Nevertheless, the definition of a deterministic rule needs some user knowledge, and an efficient definition for one optimization problem might be totally inefficient for another one; for this reason also this way doesn't resolve completely the problem.

4.5.2 Adaptive parameters' control

This approach uses heuristic rules, which take into account information about the progress achieved by the evolution process to adjust in a reasonable way the control parameters. This technique differs from the previous one because it is based on the feedback gained by the evolution, increasing the flexibility and the ability of the algorithm; in fact the magnitude and the direction of the parameter's change is the result of the evolution itself.

One approach, already presented for the ODE algorithm [30], modifies the domain of the parameters according with the current state of the evolution or of them (for ODE the domain changing was related to the variables, in order to find the current opposite); for example, the scaling factor could follow the Ali and Törn [42] rule:

$$F = \begin{cases} \max\left(F_{\min}, 1 - \left|\frac{f_{\max}}{f_{\min}}\right|\right) & if \left|\frac{f_{\max}}{f_{\min}}\right| < 1 \\ \max\left(F_{\min}, 1 - \left|\frac{f_{\min}}{f_{\max}}\right|\right) & otherwise \end{cases} \tag{4.44}$$

where f_{max} and f_{min} are the maximum and minimum values of the objective function in the current population.

This formulation reflects the demand to make the search more diversified at early stage and more intensified at latter stages: in fact, when the diversity inside the population is high, the difference between maximum and minimum function values is high and consequently the scaling factor assumes values close to 1, enhancing the exploration. The resulting scaling factor lies in a defined interval, $F \in [F_{\min}, 1]$, according with the state of the evolution.

A more complex approach is presented by Zaharie [43], after previous theoretical results [41], to adapt the control parameters according with the diversity induced in the next population, controlling the ratio between variances of the current and previous generation; taking the previous formulation about the coupled effects of the three control parameters, an adaptive scheme could be drawn:

$$2 \cdot F^2 \cdot CR - \frac{2 \cdot CR}{NP} + \frac{CR^2}{NP} + 1 = \gamma \frac{Var\left(x_{G-1}\right)}{Var\left(x_G\right)} \tag{4.45}$$

In this formulation, the right hand side is known, since the ratio between variances could be computed, and the variable γ is a user-defined parameter. In this way the algorithm undergoes a repairing effect, adapting alternatively the

scaling factor and the crossover rate (since there is an equation with two unknowns) to use in the current population *G*, according with the magnitude of the variation in diversity. The added parameter γ permits a more efficient control in the case of high increasing or decreasing; the recommended value is slightly greater than 1. Still, the Zaharie approach could be applied only on the DE strategies with some theoretical result about the evolution.

Another interesting work made in the adaptive direction is the fuzzy logic implemented to train the algorithm. This version, called FADE – Fuzzy Adaptive Differential Evolution [44] – dynamically controls *F* and *CR* using fuzzy rules based on human knowledge, giving better convergence speed to the algorithm, especially in high dimension problems.

A slightly different approach from the previous one presented is the application of competition between different DE schemes inside the same algorithm proposed by Tvrdík [45]: this idea permits the selection of the most adequate scheme; this selection is driven by the success of the scheme adopted. The competition could mix with an adaptive (or quasi-self-adaptive) approach the scheme with the most appropriate search ability according with the evolutionary stage.

Defined *H* settings (combination of *F* and *CR* or different DE schemes), the algorithm time by time adopts one of them according with the associated probability, computed as follow:

$$q_h = \frac{n_h + n_0}{\sum_{b=1}^{H} (n_b + n_0)} \tag{4.46}$$

$$\forall h \in \{1, 2, \ldots, H\}$$

where n_h is the current number of the hth setting successes and n_0 must be greater than one to prevent dramatic change in the probability. In order to avoid degeneration of some strategy, when one probability decrease below a defined value, all the probability q_h are reset to the initial value $1/H$.

The mutation process induced by a scheme is considered successful if the generated trial point shows better fitness than the target vector, that is the trial point takes place in the next generation.

The last remarkable approach presented is called SACPMDE – Self-Adapting Control Parameters Modified Differential Evolution [46] – and it combines in a greedy manner the DERL [13] approach in order to evaluate the magnitude of the perturbation induced in the scaling factor. According with the previous classification, this technique must be categorized in the adaptive control parameters. *F* is dynamically adjusted according to the relative position of the two randomly selected solutions used in the difference vector: the three

randomly chosen vectors are sorted in order to give a precise direction to the *F* perturbation. The scaling factor is then computed as follow, according with the syntax used for DERL in Section 4.2:

$$F_i = F_l + \left(F_u - F_l\right)\frac{f_{d_1} - f_{tb}}{f_{d_2} - f_{tb}} \tag{4.47}$$

This technique uses fitness information in order to weight the scaling factor: if the fitness function difference to the numerator is small, it indicates the proximity of the two solutions; otherwise, a larger scaling factor is generated, in order to explore other regions of the search space.

While the scaling factor is generated according with the tournament best approach, the crossover rate *CR* needs population information to be adapted.

$$CR_i = \begin{cases} CR_l + \left(CR_u - CR_l\right)\dfrac{f_i - f_{\min}}{f_{\max} - f_{\min}} & if\ \ f_i \geq \overline{f} \\ CR_l \quad otherwise \end{cases} \tag{4.48}$$

In that way *CR* reflects the amount of the diversity to induce in the next generation for each individual, according to the fitness function of the i^{th} solution related to the averaged state of the population: if the target vector has high fitness value, that means poor performance, the crossover rate is large, allowing the entrance of new information.

4.5.3 Self-Adaptive parameters' control

This approach represents the evolution of the evolution: the parameters to adapt are encoded into the chromosome and undergo the algorithm's operators in order to permits the survival and the propagation of the better parameters, which are more likely to produce good offspring: thereby the parameters need only an interval of existence.

The self-adaptive approach, like the adaptive, adapts only the scaling factor *F* and the crossover rate *CR*, but it takes these two parameters as variables that affect the solution.

Some self-adaptive strategies are presented below.

SACPDE and its variant

One version and its variant, called SACPDE and SACPDE2 (called also jDE and jDE2) proposed in [47] and [48] respectively, uses the following formulations for the control parameters' evolution:

$$F_{i,G+1} = \begin{cases} F_l + U_1(0,1) \cdot F_u & \text{if } U_2(0,1) < \tau_1 \\ F_{i,G} & \text{otherwise} \end{cases}$$
$$CR_{i,G+1} = \begin{cases} U_3(0,1) & \text{if } U_4(0,1) < \tau_2 \\ CR_{i,G} & \text{otherwise} \end{cases} \quad (4.49)$$

This procedure seems substituting the setting of F and CR with the setting of τ_1 and τ_2, but these two values don't show high sensitivities on the behaviour of the algorithm. They could be chosen from the interval [0.05, 0.3]. Defined these two values and the upper and lower values for the scaling factor F_u and F_l, the self-adaptive algorithm allows the propagation of the fittest individuals that bring the parameters used to generate them. This approach gives more flexibility to the algorithm, without any restriction during the evolution about the control setting. In order to make the algorithm totally flexible, SACPDE2 uses the same formulation regarding the parameters' evolution, but it implements different DE strategies (like in the adaptive DE proposed by Tvrdík [45]), which need different parameters' setting. For this reason the individual's chromosome is composed by the variables of the system and parameters assigned to each strategy used.

SaDE algorithm

SaDE [49] uses different DE strategies coupled with a different approach for the self-adaption of the parameters: it takes information from learning periods, in which the success of strategies are collected together with the CR that allow the generation of good children. The scaling factor is not adapted but just generated randomly with normal distribution within a wide range $F \sim N(0.5, 0.09)$, while $CR \sim N(CR_m, 0.01)$.

After the learning periods, new strategy probabilities and crossover rate average are computed, in order to direct the evolution toward the necessary strategy with the correct crossover rate.

SDE algorithm

A possible approach is to use the mutation rule also for the control parameters: instead the variables of the individuals, the parameters that now belong to the chromosome are generated applying the mutation process, as the following formulation for the scaling factor:

$$F_{i,G+1} = F_{r_1,G} + N(0, 0.25) \cdot \left(F_{r_2,G} - F_{r_3,G} \right) \quad (4.50)$$

The magnitude of the perturbation (in this case it is used a normal distributed scaling factor for F) slightly depends on the problem nature. This technique is

proposed in SDE [50] and SPDE [51] with some advantages, especially for noisy functions.

4.6 Multi-objective optimization

As stated in Chapter 1, the correct way to handle a multi-objective optimization process is to find a set of non-dominated solutions that form a Pareto-frontier from which take one equally good solution. However, some alternatives are used in practice for GAs and EAs in general (Section 3.3).

EAs, like DE and all its variants, have recently wide success in this practice especially for their population based-approach that allows multiple function evaluations in a single run.

The concept of non-dominated solution is quite different from a single-objective optimization: anyway the first attempt used for solving MO problem was collecting all the properties $f_k(\underline{x})$ in a unique overall function, as for the penalty method used in handling constraints. In fact the penalty method's feature is to incorporate the constraints into the objective function, penalizing it.

For their similar nature, MO problems could be solved as single-objective optimization constrained problems, choosing an objective function penalized by the others. The weakness of this method is the uniqueness of the solution, dependant on the design of the penalty function. Nevertheless, the most pursued practice is providing multiple solutions and passing the final solution to a decision maker, maybe helped by a clustering method of the Pareto-front.

In these cases the multi-objective optimization problem becomes a single-objective problem, treated by the algorithm as usual; it's important adopting a correct definition of the integrating function; the easiest way is using the weighted sum of the normalized objective functions:

$$f_{k\_norm} = \frac{f_k - f_{k\_\min}}{f_{k\_\max} - f_{k\_\min}} \tag{4.51}$$

$$z(\underline{x}) = \sum_{k=1}^{q} w_k \cdot f_{k\_norm}(\underline{x}) \tag{4.52}$$

where q is the number of the properties to optimize or the sum of the number of properties and constraints blended in the overall function, w_k are the functions' weighting factors, $0 < w_k < 1$ and usually $\sum_{k=1}^{q} w_k = 1$.

Then the optimization task (in terms of minimization) becomes:

$$\underline{x}^* \Big| z\left(\underline{x}^*\right) < z\left(\underline{x}\right) \quad \forall \underline{x} \in \Omega \tag{4.53}$$

Another example of overall function could be:

$$z\left(\underline{x}\right) = \left| \frac{1}{\sum_{k=1}^{q} w_k \cdot f_{k\_norm}\left(\underline{x}\right)} \right|^{1/q} \tag{4.54}$$

The normalization is a good practice because the objective functions could have different orders of magnitude, especially in real-world optimization process. If minimum and maximum values are not known, as in many cases of industrial processes, these values are estimated from the current population.

The choice of the weighting factors, or in general the design of the overall function, unbalances the result of the optimization, giving different importance to the objective functions. The setting of these weights move the optimization toward a specific objective function: if the weight is high respect to the others, the algorithm tends to explore the region of minimization of that target. Changing then the weights, each run returns a point that should lie on the Pareto frontier. In order to obtain a dense Pareto front, the number of weights' settings and runs must be high.

Clear examples of MO problem solved with an overall function are all the economic problems (plain aggregating approach): all the weighted factors are replaced by the costs of the properties of the system, shifting really all the objective functions under an economic point of view.

Another approach that could be implemented in EAs is a non-Pareto approach in which the total population is divided in sub-population, each of which has to optimize only one objective function: this approach, used in VEGA [7], is poor in Pareto terms, since the non-dominance of the solutions generated is limited to the reference population of the objective function.

However, better results are given by a non-dominated sorting algorithm, based on a Pareto approach, in which the sorting procedure is called after each generation to remove dominated solutions, refining the population, and ranking the remaining solutions; the idea was proposed in [1] and successively applied. The most famous GAs developed for MO optimization are: NSGA-II [52-53], SPEA [10] and PAES [11].

DE could tackle the multi-objective optimization in different ways: the classic archive approach, briefly resumed in Section 3.3 for GA, fits nicely thanks to the goodness of the mutation and crossover procedures of DE, but the improvements respect to GA for this problem are not significant. It works similarly creating a population of trial vectors, and it ranks this temporary

population, as for the progeny in GA. The trial solutions with rank 1 are then sent for comparison in the main archive.

In fact, in recent years a slightly different approach is used for DE in multi-objective optimization: the archive of non-dominated solutions is removed, and only the population is the archive present.

Common features of the Pareto-based approaches are that the Pareto-optimal solutions in each generation are assigned either the same fitness (or rank) and that some sharing or niche technique is adopted in the selection procedure.

Some way to solve multi-objective optimization with DE are then presented and briefly described, since the main feature of DE are the same for single and multi-objective problems.

PDE Approach

This method – Pareto-frontier Differential Evolution Approach [54] – uses the classical *DE random* (see Section 4.1) approach with some modification and adaption for MOP:

- The initial population is initialized according with a Gaussian distribution
- The scaling factor is normally distributed $F \sim N(0,1)$
- The individuals used for reproduction must be a non-dominated solutions
- Some repair rule referred to the domain is applied
- Trials replace their basis vectors if they dominates them, otherwise the reproduction is repeated
- All the dominated solutions are removed
- If the number of non-dominated solutions exceeds some threshold, a distance metric relation is used to remove solutions close to each others.

This method is very sensitive to the *CR*, and it works better with low crossover rates, evident sign of low convergence speed of this method. Nevertheless the resultant Pareto-frontier has good diversity. A Self-adapting approach on *CR* and mutation rate, inherited from the parents, is combined with this method: the new algorithm, SPDE [55], presents improved behaviour, convergence speed and superiority compared with other algorithms.

MODE algorithm and its variants

The first proposed MODE – Multi-Objective Differential Evolution [56] – is practically similar to PDE, with some little difference about the initialization, the handling constraints and the removal of crowdedness, since its first application was the optimization of an industrial process.

Based on real-optimization problem, the initialization is performed uniformly, a penalty method is applied for handling constraints and the number of

population decreases in every generation because if a child doesn't dominate its target vector, the reproduction is not repeated as in PDE. In fact the first version of MODE represents just the application of non-dominating sorting in DE to skim the population, removing dominated solutions and achieving only the non-dominated ones to continue the reproduction. The comparison with other algorithm was based on economic evaluation and the results were interesting. One of the main weaknesses of this implementation is the fast diminishing of the individuals in the population: applying a removing of dominated solutions each generation, the population size quickly diminishes, loosing in diversity. The reproduction procedure has poor genetic material to mix and the stagnation is achieved soon. For this reason, this multi-objective scheme is considered unsatisfactory.

Other modifications [50] are then introduced to overcome the clear lacks of the previous version of MODE: a second version, MODE-II, maintains the number of individuals in the population constant, generating random solutions, even if dominated, after the removing of dominated solutions for each generation. In that way the algorithm has more probability to continue the evolution without any premature stagnation due to the diminishing of the population size, since the constant insertion of new genetic material. This approach works better respect to MODE-I, achieving the Pareto front, but the time to obtain a solution is comparable or even higher respect to the other algorithms.

A third version, MODE-III, uses a revolutionary idea for the multi-objective optimization: it exalts the recombination of DE and its selection procedure, applying the removing of dominated solutions only at the end of the evolution.

In this scheme, each trial vector, generated by mutation and crossover operations, is compared only with its target from which it inherits some variable, and, if the trial dominates the target vector, it takes its place in the population for the next generation, otherwise the target vector survives. The selection is therefore applied with its original purpose but in multi-objective concept of dominance. Unexpectedly, this procedure works well, saving considerable time because no ranking of the population is adopted during the evolution and a dominated comparison is made only *NP* times each generation (comparison in the selection process between trial and target). Of course, without any ranking, the selection of the individuals for reproduction can be only random, since no fitness information could be used for the selection of parents without a ranking of the population. Thanks to the goodness of the reproduction ability and exploration of DE, the Pareto frontier is achieved by a high fraction of the population.

As for the previous MODE versions, MODE-III does not use archive for the storage of non-dominated solutions. The only archive present is the population: it starts at the beginning with few non-dominated solutions, but the greediness of the selection procedure for the next generation is its strength. In that way,

the archive of the non-dominated solutions and the population are the same thing.

Some comparison is made in [57] and it is clear, MODE-II and MODE-III outperform MODE in terms of Pareto-frontier's shape at the cost of extra computational time. MODE-III, anyway, is considered the most reliable and promising DE variants for multi-objective optimization.

Successive improvements, hybridizations and sophistications are proposed in literature; the goodness of these attempts is clear since DE gains a lot from the blending of techniques and methods.

Some interesting examples are the H-MODE proposed in [14], [15], where each non-dominated solution is then locally optimized by s sequential simplex method, and the application of trigonometric mutant operator to MODE-III proposed in [58].

Chapter 5

Case studies

This chapter has the purpose to show the improvement of DE respect GA, both in single-objective and in multi-objective optimization. Farther, DE is tested on real optimization cases of complex industrial systems for the Oil&Gas industry and the nuclear industry.

The comparison is made first with benchmark problems, characterized by different dimensionalities and complexity, in order to evaluate the behaviours of the algorithms and the sensitivities of DE on its parameters.

DE is then applied to a real case of the Oil&Gas industry: thanks to the apprenticeship made inside the PROD department of Eni E&P division, an integrated optimization tool, equipped with DE, has been built. This tool is flexible and adaptable to many situations. Its general task is to optimize whatever property of the system defined by the user. A particular highly-constrained case is taken as case study for the goodness of the tool.

At the end, DE is tested on a reference case for the nuclear industry: the inspection intervals optimization is a difficult task for a safety system, since the presence of conflicting objective functions. Then, the problem is tackled with a multi-objective optimization. Starting from results previously obtained on this problem, the DE abilities and results are compared with GAs suited for multi-objective optimization.

All the results present in this chapter are obtained on a machine with these characteristic:

HP, Genuine Intel® CPU T2050 @ 1.60 GHz, 0.99GB of RAM.

5.1 Comparison in single-objective and multi-objective optimization on benchmark problems

The comparison is made with the purpose to demonstrate the robustness and the high reliability of DE and many of its variants; the improvements respect to GAs are measured in different ways, since the different optimization's natures. The tests reported in this section are conduit on benchmark problems taken from literature, even for single-objective and multi-objective optimization. A final conclusion is made at the end of the tests.

The evolutionary algorithms treated in this chapter are:

1. *Genetic Algorithm Toolbox* developed by Mathworks; this a commercial version for GA, suitable for several problems without any re-programming phase, offered by Mathworks; the setting is not banal, good solutions in complex problems could be achieved only with a correct setting of the whole sophistications after a previous tuning;
2. *Multi-Objective Genetic Algorithm MOGA*, a tool developed in FORTRAN by LASAR (LAboratory of Signal and Risk Analysis http://lasar.cesnef.polimi.it/) of the Energy Depratment of Politecnico di Milano; it has several variants adoptable, both in single-objective and multi-objective optimization, and the number of information necessary to its running is high. Furthermore, a wrong strategy selection could provoke failure of the optimization. Also for this tool the setting is not easy.
3. *Multi-Objective Differential Evolution MODE*, developed by LASAR, provided with the single-objective and multi-objective optimization options. Several variants are implemented in the tool, in order to increase its flexibility and ability to tackle different problems. For multi-objective optimization option *MODE-III*, described in Chapter 4.6, is implemented;
4. *Simple real-coded GA*: it has an easy implementation of GA written in Matlab; the structure is practically the basic version of GA, without any further specific alteration. The setting is easy but the reliability is poor in complex situations.

5.1.1 Single-objective optimization

This section is organized as follows: the problem is first briefly described, then, the setting for any algorithm is explained. Then, the results and some sensitivities are reported and commented, taking into account the characteristics of the algorithms and the setting adopted.

The problem

This case study is conduit on 23 benchmark functions taken from [59] and reported in Appendix A. The functions have different properties, dimensionalities and complexities. The true global optimum of the objectives functions are known and usually clearly defined or defined with a good accuracy (maximum error = $1 \cdot 10^{-4}$).

For this case study the optimization is unconstrained; then, no methods for the satisfaction of constraints are reported; only repair rules for the satisfaction of the solution's existence on the domain are applied.

The selection of these benchmark functions has the aim to understand the behaviour of different variants of algorithm, since it is clear the reproduction

method, the parameter setting and the stopping criteria influence the issues of the optimization.

The dimensionalities are between 2 and 10, the domain could be wider, in order to evaluate the speed of the algorithm to restrict the searching area, or the function shape could be multimodal, to evaluate the ability of avoiding local minima (several or close to the true optimum).

In this case study the algorithms *GA-toolbox*, *MODE* and *simple GA* are tested.

Each optimization is repeated 50 times in order to obtain significant statistical values with respect to the randomness.

The algorithms' setting

Genetic Algorithm toolbox (*GA-toolbox*) has several sophistications and internal variants. A complete descriptive help is available on the program and online. When no particular settings are imposed to this tool, many of the sophistications implemented are used with default setting. Anyway, the correct usage for a specific problem needs substantial knowledge of the tool.

The *GA-toolbox* setting is made by literature and owner recommendations; the options applicable to this tool are several, but for our test the setting is restricted to basic options like population size, selection rule, crossover rate and replacement procedure.

The whole of these options are explained in the help of the function `gaoptimset`. When nothing is specified, the tool sets automatically the default value recommended by the owner.

The setting used in our test is:

```
'PopInitRange'          [low;up]
'PopulationSize'        30
'EliteCount'            dim
'CrossoverFraction'     0.7
'Generation'            500
'TolFun'                1e-4/1e-8
'StallGenLimit'         50
```

The default setting for the parents' selection is the so called, `Stochastic uniform`: it lays out a line in which each parent corresponds to a section of the line of length proportional to its scaled value. The algorithm moves along the line in steps of equal size. At each step, the algorithm allocates a parent from the section it lands on. The first step is a uniform random number less than the step size.

The crossover is single-point and it is applied with probability 0.7, defined by `Crossover-Fraction`.

The replacement rule for the next generation is the simplest one already presented in Section 3.2: the two new children generated replaces the parents. Only this option is allowable in the tool.

The values `low`, `up` and `dim` are different for each benchmark function and loaded function by function. The population and the maximum number of generation are fixed for all the algorithms respectively to 30 and 500, in order to have a fixed maximum number of function evaluations as 15000.

The option `EliteCount` specifies the number of best solutions that survive to next generation without any change, and this value is set as the dimensionality of the problem.

A default option for the mutation uses the classic uniform mutation with probability `0.01`.

The stopping criteria adopted are two:

- `StallGenLimit` generations over which cumulative change in fitness function value is less than `TolFun`
- reached `500` generations

In order to test the ability of this tool and reach similar behaviour with the other algorithms, the value of `TolFun` is diminished till `1e-8`.

For *MODE* in single-objective optimization, eleven variants are implemented and tested in order to evaluate the goodness of each strategy. These variants are seven promising mutation variants described in Section 4.1 and 4.2 and four adaptive or self-adaptive schemes explained in Section 4.6. They are considered, efficient, easy to use and reliable.

Further sophistications, like *ODE*, *DELB*, *DEPC*, *MDE* and *NSDE* (Nonlinear Simplex DE), don't belong to the class of basic modifications on solutions and they are not tested, because their improvements are independent and applicable regardless the mutation scheme adopted. They are considered hybridization of the optimization process between different strategies, like for *NSDE* or *ODE*: the skills of Nonlinear Simplex Method or of the Opposition Based Optimization are coupled with the robustness and reliability of DE.

The proposed implementations have approximately the same parameters (except for the adaptive schemes) but different mutation approaches, taking sometimes information from the fitness.
The following common setting is adopted:

```
Population size NP      30
MAXGEN                  500
eps                     1e-4
```

and the diversified settings are:

1.	*DE random*	`F=0.5, CR=0.5`
2.	*DE best*	`F=0.5, CR=0.5`
3.	*DE current-to-best*	`F1=0.8, F2=0.5, CR=0.5`
4.	*TDE*	`F=0.5, CR=0.5, MT=0.1`
5.	*NSDE*	`CR=0.5, NS=0.5`
6.	*DERL*	`F=0.5, CR=0.5`
7.	*DERL 2*	`F=0.5, CR=0.5`
8.	*DE_adapt*	`Fmin=0.1, CR=0.5`
9.	*SACPDE*	`Fmin=0.1, Fmax=1, Fc=0.1, CRc=0.1`
10.	*SACPMDE*	`Fmin=0.1, Fmax=1, CRmin=0.05, CRmax=0.8`
11.	*SDE*	`OPmean=0, OPstd=0.7`

The crossover rate *CR* is set as 0.5 for all the variants. This choice is driven by the necessity to test the different strategies under similar algorithm's conditions but over several problems. As for *CR*, also the scaling factor *F* is set as recommended by the literature. Since the correct working range of *F* is (0, 1], instead the initial definition of (0,2] made by Storn and Price [21], this parameter is set to 0.5 if the variant requires it. In the other cases, like for *DE current-to-best*, the second scaling factor F_2 has the same role of the classic scaling factor; then it is set to 0.5, while the first scaling factor F_1 is set to 0.8 in order restrict the searching area. For *TDE*, since it uses also the *DE random* reproduction technique, the set of *CR* and *F* is as the other, while the probability of trigonometric reproduction is set as 0.1; so, the 10% of the mutation phase is performed with *TDE*. This set should show the goodness of this practice over the 23 benchmark problems. *NSDE* has like the others 0.5 as crossover rate; the parameter $NS \in [0,1]$ control the step-length of the search: if *NS* is high, the search is more concentrated in the neighbourhood, whereas if it is small the step length is high, useful for exploration on large domains. In order to keep balance between this two strategies, *NS* is set to 0.5.

For the adaptive and self-adaptive variants the parameter setting is necessary but it's less sensitive on the final result, since a domain for the parameters is required: for all of them, F_{min} is equal to 0.1, while for *SACPDE* and *SACPMDE* also the maximum scaling factor is required: F_{max}=1. *SACPDE* (see Section 4.5.3) requires also the two added parameters, τ_1 and τ_2, here called *Fc* and *CRc*. They control the probability of the evolution of the parameters *F* and *CR*. They are set as 0.1, a value recommended in [47-48], since frequent parameters changes are not beneficial for the evolution. *SACPMDE* needs minimum and maximum values for crossover rate: 0.05 and 0.8 are considered the maximum and minimum value for good algorithm behaviour. The evolution of the

parameters in this case is driven by fitness feedback information. *SDE* has the two parameters OPmean and OPstd: its mutation procedure applied to the parameters *F* and *CR* uses a Gaussian random variable with mean equal to OPmean and standard deviation equal to OPstd. In order to obtain different scaling factor for this procedure, even negative, the mean is set as 0.

The stopping criteria for *MODE* are:

- $\Delta=|f_{min}-f_{max}|$ of the current population is less than eps: the whole population is converged at the same point if eps is sufficiently small respect to the fitness's order of magnitude;
- reached MAXGEN generation.

When the first stopping criterion is met the algorithm alts because no more exploration could be performed. In case of multimodal function with local optima, the alt by the first criterion to a wrong solution means the inability of the algorithm to find true optimum.

The *simple real-coded GA*, implemented with the purpose to show the feature of a basic GA, is written in one script in Matlab. Since *DE random* has an easy implementation as *simple GA*, a direct comparison could be done between these two EAs.

DE random represents the basic idea of DE, while *simple GA* has the classical procedures of GA for real-coded variables.

The encoding for *simple GA* is made in floating-point representation. The parents' selection procedure is called *Fit-Random Selection*, and it is hybridization between the Fit-Weak Selection and the Random Selection explained in Chapter 3; after the ranking of the population by fitness comparison, the first parent is selected from a fittest fraction of the population defined by the user: the lower this fraction, the higher the fitness of the first parent. Nevertheless, if this fraction is too low, the number of solutions at selection disposal is too small and the evolution could be affected by premature stagnation. The second parent is then selected randomly from the entire population. On average, the fitness of the second parent is lower respect to the fitness of the first one, as for the Fit-Weak Selection, but the randomness introduced for the second parent selection leaves interesting opportunities to avoid the weaknesses of the two selection procedure hybridized. The crossover method is single-site crossover, and it is coupled with the arithmetic blending rule (3.2). The replacement rule is a Fittest Replacement but applied to the pool formed by the parents and children populations. This replacement is coupled with a high random mutation probability to avoid stagnation.

The parameters' setting for *simple GA* is:

```
Population size NP      30
Fit selection fraction  NP/2
```

`Reproduction alfa`	`0.7`
`MAXGEN`	`500`
`eps`	`1e-4`
`Mutation Mt`	`0.1`

As for *GA-toolbox*, the population size and maximum number of generation are 30 and 500 respectively. The hybrid selection procedure chose the first parent from the `fit selection fraction`. The `alfa` parameter is the fraction inherited from the first child to the presumed fittest parent used in the blending method: the second child inherits the reciprocal genetic material from parents.

The stopping criteria are the same as for *MODE*.

Measures

The measures utilized to evaluate and compare the goodness of the algorithms are taken from literature and allow a correct characterization of the solutions. These measures, proposed for SO, have statistical significance since are reported as average over the 50 runs:

1. Function evaluation fe: this value represent the number of the objective function's calling; higher the fe, slower the convergence of the algorithm. Anyway, since the stopping criteria are referred to a maximum value for generations (and consequently to fe) or the maximum fitness difference inside the population, fe equal to the maximum value (`NP*MAXGEN`) doesn't indicate the absolute inability of the algorithm, because the true optimum could be reached by some solution in the algorithm and not by the whole population.
2. Cpu: it indicates the cpu time (expressed in seconds) necessary to complete the optimization. Together with fe, it could represent the convergence of the algorithm but also the complexity of the implementation.
3. Success rate sr: it expresses in % the fraction of success to find the optimum (it isn't an average); it represent the ability to find the true optimum under a specified tolerance (eps=1e-4). If the optimum found by the algorithm is closer with a smaller tolerance than eps, the optimization is considered with success. It could be computed only if the true optimum is known. The success could be achieved even if the number of function evaluation reaches the maximum: that means not the whole population converge to the same point but at least the best solution is locate to the optimum.
4. Relative error lambda: this value, already proposed in [45], is useful to compare the accuracy of a solution. The higher the lambda, the higher the accuracy of the solution. The value is referred to the fitness function value and the rule is:

$$if\ c \neq 0\ \ \lambda = \begin{cases} 0 & if\ \dfrac{|m-c|}{|c|} \geq 1 \\ 11 & if\ \dfrac{|m-c|}{|c|} < 1 \cdot 10^{-11} \\ -\log_{10}\left(\dfrac{|m-c|}{|c|}\right) & otherwise \end{cases} \qquad (5.1)$$

$$if\ \ c = 0\ \ \lambda = \begin{cases} 0 & if\ |m| \geq 1 \\ 11 & if\ |m| < 1 \cdot 10^{-11} \\ -\log_{10}\left(|m|\right) & otherwise \end{cases}$$

Where *m* is the value found by the algorithm and *c* is the certified true optimum.

For a complete evaluation on the 23 benchmark functions, the summed values of the previous measures are used as comparison between strategies. Of course the sum cannot explain deeply the behaviour but gives a meaningful overview.

Results and sensitivities

Table 1 reports the results obtained by the two type of GA tested: *GA-toolbox* and *simple GA*. For the first, two eps (1e-4 and 1e-8) as stopping criteria are used and reported.

Figures 6, 7, 8 and 9 plot the summed values over the 23 problems of the fourth measures (fe, cpu, sr and lambda in sequence) for the three test on GA and for the results obtained with the basic DE variant *DE random*, reported in the first column of Table 4 together with other DE variants.

Simple GA has the worse behaviour even in comparison with *GA-toolbox*. The number of fe is often high (Figure 6), close to the maximum for the more difficult problems that have dimensionality over two (f5, f6, f8, f15, f19-22, see Appendix A for further information). As said before, high fe does not mean failure of the run, since the true optimum could be achieved by some chromosome in the population. In fact, sr has values between 0 and 92 for these functions (Table 1): that results depend on the problem nature and the specific exploration ability of the algorithm, which could be effective for some function's shape (e.g. f8, f15) and completely inefficient for another one. Anyway, from a general point of view, *simple GA* uses a significantly higher number of fe, with a scarce sr and the lowest lambda in the tests (Figure 8 and 9), that means small accuracy of the solutions.

Analysing the *GA-toolbox* results, a first oddity is the number of fe, really small, even with TolFun=1e-8; this behaviour is due to the different stopping criteria allowable: in fact, many of the runs stop at the value 1560 fe, that means

52 generations. Only after 2 generations the cumulative change in fitness function is less than Tolfun, sign of a scarce diversity in the population, even with a crossover fraction of 0.7. This premature stagnation however does not imply low sr, since this measure is often a good value, but it means high exploration in early stages and rapid loss of diversity, which could be symptom of ineffectiveness for some specific problems. The lambda values are acceptable but the cpu, especially for the smallest TolFun, is almost the double (Figure 7). This behaviour depends on the implementation: the *GA-toolbox* has many functions and scripts callings due to its complexity, while *simple GA*, written in a single script, has nearly four times fe and less cpu.

Comparing the two *GA-toolbox* tests, a diminishing of TolFun increases as expected the performances: sr and lambda increases, together with fe and cpu, since no greedy alterations are introduced: only a more strict stopping criterion is adopted and the searching is obliged to continue.

Table 5.2, *GA-toolbox* and *simple GA* results on the 23 benchmark functions for SO. *GA-toolbox* is tested with two eps values (1e-4 and 1e-8).

		ga-toolbox eps=1e-4				ga-toolbox eps=1e-8				simple_ga			
f	N	fe	Cpu	sr	lambda	fe	Cpu	sr	lambda	fe	cpu	sr	lambda
1	2	1560	0.262188	48	3.337935	1560	0.260625	70	4.429559	9359.4	0.201563	14	3.666839
2	2	1560	0.255938	2	1.782227	9119.4	1.466875	10	3.066425	9305.4	0.19875	74	4.937371
3	2	1560	0.277813	100	5.294264	1560	0.289688	98	5.193207	8529	0.197188	34	3.910006
4	2	1560	0.259688	100	5.732996	1560	0.260938	100	5.727077	6010.8	0.13375	36	3.95401
5	4	1560	0.257188	14	3.272079	1571.4	0.26125	8	3.518582	14439	0.386875	40	6.476024
6	10	1655.4	0.249375	0	0.464211	5467.8	0.79375	4	1.142977	14434.8	0.565	0	1.340839
7	2	1560	0.258125	56	4.702975	3231	0.525938	42	4.042097	8140.8	0.171563	60	4.144548
8	10	1560	0.231563	0	2.046865	3520.8	0.517188	26	3.597903	12152.4	0.454375	92	5.197668
9	2	1560	0.260938	86	3.140897	1560	0.2625	92	3.342827	6666	0.143125	34	2.992255
10	2	1560	0.255938	100	9.278239	1560	0.256875	100	9.188656	4248	0.089375	76	5.270968
11	2	1560	0.256563	88	6.684636	1562.4	0.257813	90	7.00122	10475.4	0.221563	34	3.401784
12	2	1560	0.258125	74	6.024265	1563	0.260938	90	7.20684	6503.4	0.137188	74	4.541488
13	2	1560	0.259063	92	8.39535	1560	0.26375	96	8.756039	3133.2	0.067813	82	4.849783
14	2	1560	0.259375	100	4.559536	1560	0.260313	98	4.470347	8883	0.19625	26	3.319797
15	4	1560	0.257813	62	3.502271	1919.4	0.318438	88	4.617155	13677.6	0.350625	98	4.998141
16	2	1565.4	0.259375	54	7.783416	1602	0.26625	56	8.368153	14670	0.314688	2	2.384518
17	2	1560	0.257188	98	8.501827	1560	0.260625	84	7.150096	4440	0.094688	54	3.6255
18	2	1560	0.257188	94	6.588132	1593.6	0.263125	84	6.058933	9528	0.203125	16	2.439503
19	10	1587	0.24125	0	1.554397	4684.2	0.690313	36	3.681347	14808.6	0.578438	0	0.750079
20	3	1599	0.264375	58	4.205901	2013.6	0.330313	98	6.025114	15000	0.3525	0	0.833544
21	3	1560	0.260625	84	5.888504	1636.8	0.274375	94	6.052147	14191.8	0.343125	84	5.089597
22	10	1832.4	0.271563	0	0	6420.6	0.926563	0	0.073786	14720.4	0.575313	4	1.346921
23	2	1560	0.2625	18	1.657683	1560	0.264375	14	1.654446	7485.6	0.165938	36	3.665762
	sum	**36319.2**	**5.93375**	**1328**	**104.3986**	**59946**	**9.53281**	**1478**	**114.3649**	**230802.6**	**6.14281**	**970**	**83.13695**

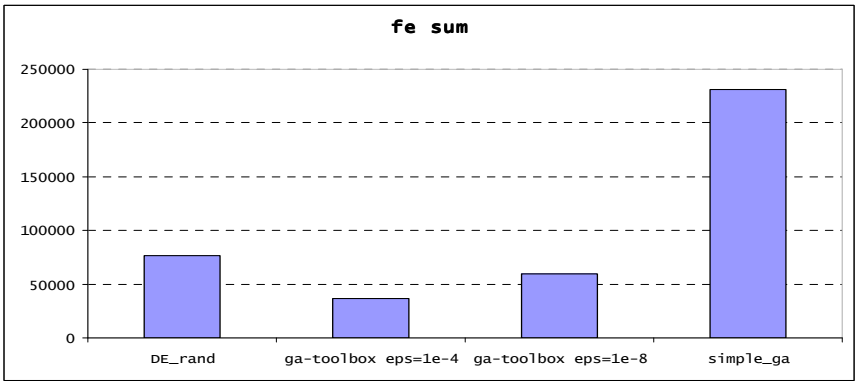


Figure 5.16, Sum of the function evaluations for three GA tested and for *DE random* over 23 SO problems.

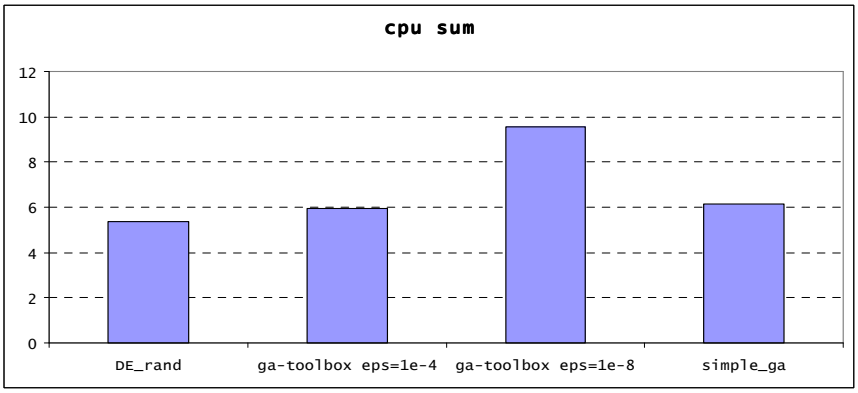


Figure 5.17, Sum of the cputime used for three GA tested and for *DE random* over 23 SO problems

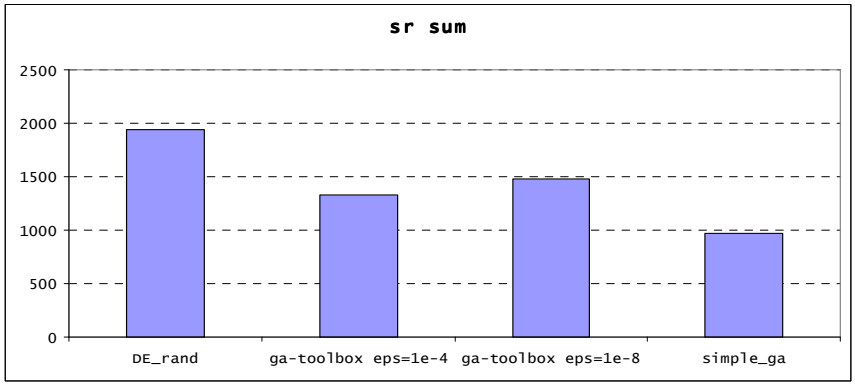


Figure 5.18, Sum of the success rates for three GA tested and for *DE random* over 23 SO problems.

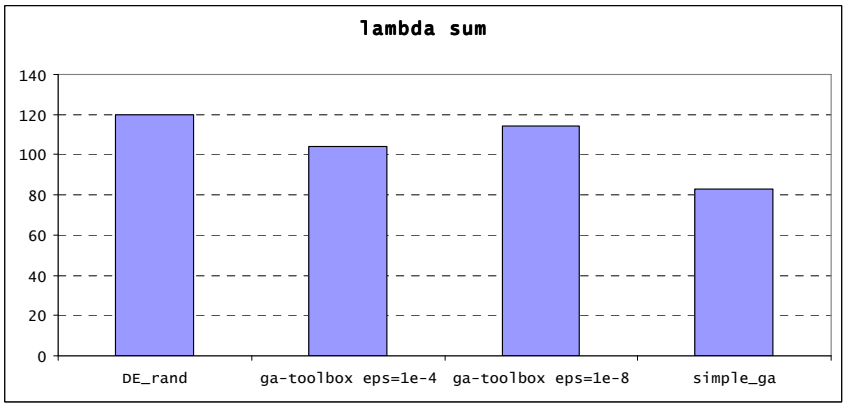


Figure 5.19, Sum of the lambda obtained for three GA tested and for *DE random* over 23 SO problems.

The basic DE version, *DE random*, (see Table 2), used as reference for DE in this test, outperforms the other algorithms in terms of cputime, sr and lambda, using however more fe than *GA-toolbox* (Figure 7, 8 and 9). The ability of DE is significantly better in almost all the problems, achieving 100% of sr, except for high dimensionality (f6, f10, f22), where the sr collapses close or to 0. This stagnation depends on the setting parameters; in fact, the three parameters could affect the optimization results. In order to find the most significant parameter for the success of DE, Figure 10, 11 and 12 show the dependencies of the results on *NP*, *F* and *CR* respectively, applied on f6, the Ackley's problem (see Appendix A), one of the most difficult since its dimensionality (*n*=10) and high number of local optima. The basic setting is NP=30, CR=0.5 and F=0.5; when a parameter's sensitivity is evaluate, the other parameters are kept as just defined.

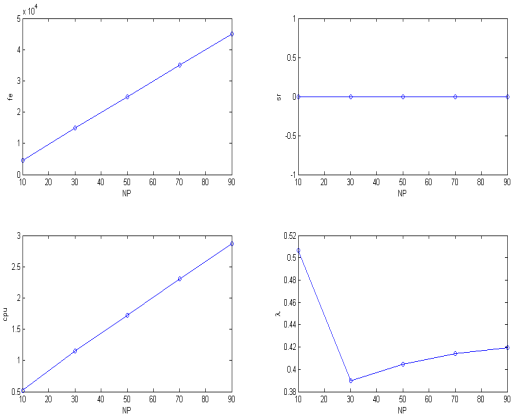


Figure 5.20, Population size's (*NP*) effect on the four measures for the Ackley's problem (f6) for *DE random* with *F*=0.5 and *CR*=0.5.

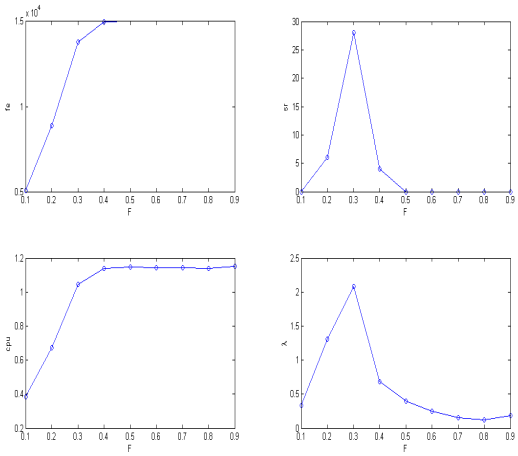


Figure 5.21, Scaling factor's (*F*) effect on the four measures for the Ackley's problem (f6) for *DE random* with *NP*=30 and *CR*=0.5.

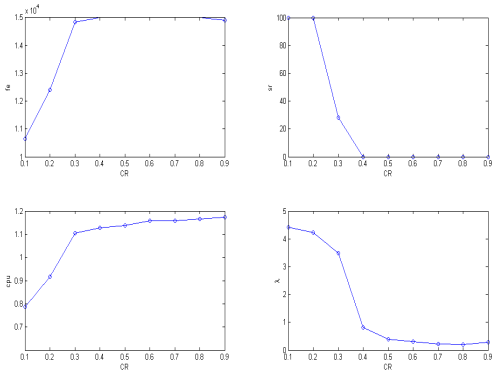


Figure 5.22, Crossover rate's (*CR*) effect on the four measures on the Ackley's problem (f6) for *DE random* with *NP*=30 and *F*=0.5.

The amount of induced perturbation is essential to find the true optimum: for this reason, the population size *NP* does not have effect on the sr and the whole runs continue till the maximum number of generation allowable: the unique effect is to increase the fe since the increasing of NP.

F and *CR* intervene on the perturbation generation by generation: *F* with the current setting has any effect only around 0.3 (see Figure 11), but not enough significant. Small *F* brings to premature stagnation since the low exploration ability and high *F* makes impossible to localize the restricted area of the true optimum.

High sensitivity is noticed with *CR* (see Figure 12): when it approaches values around 0.1÷0.2, the sr grows up till 100%, and the fe diminishes significantly. This behaviour is due to the high dimensionality and the multimodality of the function: inducing frequent modifications in the

population, premature stagnation has been found, avoiding any possibility to escape from local optima; the indicator of this situation is lambda, which settles around 0.4 for *CR* values greater than 0.5; the same lambda value is approached by settings with *F* greater than 0.5.
The effect of *F* with a good *CR=0.1* setting is showed by Figure 13.

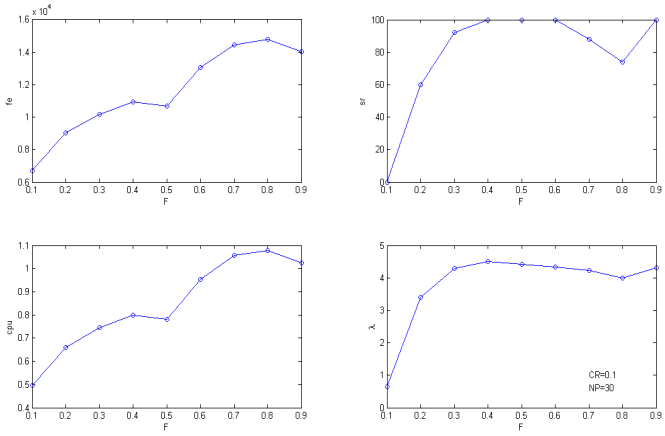


Figure 5.23, Scaling factor's (*F*) effect on the four measures for the Ackley's problem (f6) for *DE random* with *CR=0.1*.

Interesting is the behaviour of `fe` in this case: the `F` value of `0.5` demarks a trend's change on `fe`, even if `sr` remains high (over 70%) and `lambda` moves around `4` (that means success of the run, since the `sr` criterion).
It is clear, the issue of a run depends both from *F* and *CR* together, but in this case the *CR* effect is preponderant; in order to prove it in this particular case (f6, Ackley's problem), the following test is performed on *DE random* with $F_i \sim U(0,1)$, for each solution, eliminating the direct interaction between them, and three settings of *CR*, 0.1, 0.5 and ~*U*(0,1).

Table 5.3, Function evaluations, cputime, success rate and lambda obtained on Ackley's problem (f6) by different settings on *DE random*. In this case $F_i \sim U(0,1)$.

CR	fe	cpu	sr	lambda
0.1	9958	0.734	98	4.375
0.5	10892	0.825	46	2.873
~U(0,1)	10488	0.800	66	3.293

Making *CR* a uniform random variable doesn't assure high success. The results obtained with low crossover rates show an increased success rate (really close to 100%) for this difficult problem. A side effect of this set is the increased computation time due to the increased number of function evaluation necessary. Low *CR* means less improvement due to the mutation process between two

consecutive generations: however, this set allows the attainment of the global optimum. In fact, the lambda value is over 4 (the minimum eps allowable for stopping criterion is 1e-4).

Proved the importance of *CR* on the success of *DE random* on high dimensional function, the effects on the whole 23 functions of this parameter is tested for *DE random* and *DE best* with four *CR* settings: 0.1, 0.3, 0.5, 0.7.

Table 3 shows only the effects on the summed measures for the test on 23 problems.

With small *CR*, fe increases (and quasi-proportionally as cpu) in both variants, since the small perturbation induced in the population as proved with the sensitivities: the convergence speed is low but the success becomes close to the maximum; the lambda is high, symptom of high accuracy. The success for the random version is higher than the best variant, but the latter uses less fe and at the same time it has better accuracy: that means *DE random* is more reliable but with a lower convergence speed and a lower general accuracy.

Table 5.4, Summed measures for different *CR* settings used in the test on 23 problems for two variants: *DE random* and *DE best*.

Set	Σfe	Σcpu	Σsr	Σlambda
DE random				
CR=0.1	84253	5.600	2222	140.9
CR=0.3	78219	5.328	2104	131.9
CR=0.5	76865	5.370	1938	119.8
CR=0.7	75156	5.339	1938	117.5
DE best				
CR=0.1	54168	3.698	2094	142.9
CR=0.3	37164	2.540	1934	125.2
CR=0.5	29894	2.049	1886	120.6
CR=0.7	23770	1.631	1810	114.4

This trend could be expected for all the variants proposed but surely with different sensitivities. For this reason the usage of a unique *CR*=0.5 in the evaluation of the other variants permits the understanding of the single abilities, without taking into account the most significant impact due to *CR* just proved.

Tables 4-7 show the results in the 23 functions for the eleven variants proposed and Figures 5.9-5.12 plot the summarized results.

Table 5.5, Results on 23 benchmark problems with different dimensionality and complexities for *DE random*, *DE best* and *De current to best* variants.

		DE random NP=30, CR=0.5, F=0.5				DE best NP=30, CR=0.5, F=0.5				DE ctb NP=30, CR=0.5, F=0.5			
f	n	fe	cpu	sr	lambda	fe	cpu	sr	lambda	fe	cpu	Sr	lambda
1	2	1388.4	0.08625	100	5.955095	725.4	0.046563	100	5.949586	15000	0.909375	96	5.736093
2	2	1861.8	0.110313	100	6.587912	870.6	0.05125	100	6.86819	11634.6	0.69875	100	10.0571
3	2	1084.2	0.065625	100	5.301616	565.2	0.034063	94	4.995698	1242	0.07625	96	5.095229
4	2	1009.8	0.06	100	5.791827	587.4	0.034688	100	5.754329	5221.8	0.323125	100	5.769498
5	4	2529.6	0.162813	100	7.108407	1176.6	0.07625	62	5.233969	13935.6	0.912813	84	6.368791
6	10	15000	1.13625	0	0.385665	5706.6	0.436875	0	0.500469	15000	1.161563	2	0.578558
7	2	2214	0.130313	64	4.98297	915	0.054063	44	4.295962	13029.6	0.782813	52	6.378667
8	10	2886.6	0.2125	100	4.668029	1229.4	0.091875	100	4.758256	1199.4	0.089375	100	4.71573
9	2	1037.4	0.060938	100	3.614112	587.4	0.034375	98	3.546022	624	0.037188	100	3.613439
10	2	1731.6	0.102188	100	6.817949	622.2	0.03625	100	7.029021	14712.6	0.885	100	10.91538
11	2	1425.6	0.083438	100	6.695279	826.2	0.04875	100	6.94955	876	0.051875	100	6.902585
12	2	1272.6	0.075	100	6.793135	694.8	0.04	100	6.822651	754.2	0.044688	100	6.779006
13	2	1070.4	0.062813	100	6.656401	607.8	0.035938	100	6.895562	645	0.038438	100	6.804108
14	2	1519.2	0.09125	100	4.567164	628.2	0.0375	100	4.564657	10987.8	0.670938	100	4.56006
15	4	1704.6	0.1075	100	5.093219	880.8	0.05625	100	5.504226	907.8	0.05875	98	5.345643
16	2	2092.2	0.122188	100	8.771529	970.2	0.056875	100	8.771251	3561	0.214688	100	8.770008
17	2	858	0.050313	66	4.462375	612	0.035625	88	6.066943	1796.4	0.107813	84	6.24475
18	2	1434	0.084688	100	7.070044	761.4	0.045	100	7.27668	817.8	0.048438	100	7.335954
19	10	14871	1.13125	8	1.627681	3836.4	0.293438	0	1.203255	14273.4	1.093125	2	1.3186
20	3	1775.4	0.108125	100	5.799182	949.8	0.058438	100	6.198577	1009.2	0.0625	100	6.203972
21	3	1456.8	0.090625	100	5.848219	778.2	0.048125	100	6.248212	11034	0.715	100	9.699106
22	10	15000	1.138438	0	0	4575	0.349375	2	0.089809	15000	1.149375	0	2.65E-05
23	2	1642.2	0.0975	100	5.17316	787.8	0.047188	98	5.083882	15000	0.914688	100	5.164255
	sum	**76865.4**	**5.370313**	**1938**	**119.771**	**29894.4**	**2.04875**	**1886**	**120.6068**	**168262.2**	**11.04656**	**1914**	**134.3566**

Table 6.5, Results on 23 benchmark problems with different dimensionalities and complexities for *DERL, DERL 2* and *NSDE* variants.

		DERL				DERL2				NSDE			
		NP=30, CR=0.5, F=0.5				NP=30, CR=0.5, F=0.5				NP=30, CR=0.5, NS=0.5			
f	n	fe	cpu	sr	lambda	fe	cpu	sr	lambda	fe	cpu	Sr	lambda
1	2	1047	0.100938	100	5.952466	1186.8	0.090625	92	5.639169	1524.6	0.10125	100	5.952198
2	2	1420.8	0.135938	100	6.737801	1274.4	0.094063	100	6.764127	2128.2	0.141563	100	6.470531
3	2	808.8	0.075938	100	5.297873	889.8	0.065625	100	5.304072	1254	0.082813	100	5.299325
4	2	819.6	0.077188	100	5.777889	977.4	0.071875	100	5.805128	1171.8	0.075938	100	5.811614
5	4	1983.6	0.198125	100	7.118401	2592	0.205313	78	6.017685	2719.2	0.190938	100	7.101337
6	10	14958.6	1.68125	4	0.759905	12654.6	1.148438	0	0.399546	13736.4	1.120625	48	3.27163
7	2	1516.2	0.142188	52	4.476547	1314.6	0.09875	50	4.323771	2504.4	0.16125	64	4.889994
8	10	2125.2	0.23375	100	4.669995	5025	0.44625	64	3.75503	3580.2	0.284375	100	4.646072
9	2	810	0.075313	100	3.614015	846	0.061875	98	3.547684	1216.2	0.077813	100	3.613859
10	2	1227.6	0.115625	100	6.851423	935.4	0.067188	100	6.381081	1948.8	0.124375	100	6.946709
11	2	1118.4	0.105313	100	6.633737	1083.6	0.079688	100	6.445334	1674.6	0.108125	100	6.746244
12	2	957.6	0.09	100	6.635263	900.6	0.065625	100	6.335837	1486.2	0.09625	100	6.623999
13	2	838.8	0.079375	100	6.702062	672.6	0.049063	100	6.485334	1258.2	0.085625	100	6.505122
14	2	1052.4	0.099688	100	4.566871	1084.2	0.080313	100	4.571627	1675.2	0.109688	100	4.568161
15	4	1320.6	0.131563	100	5.050855	1249.2	0.098125	98	4.758916	2045.4	0.143438	100	5.052771
16	2	1502.4	0.1425	100	8.77177	1687.8	0.125625	100	8.777342	2415	0.156563	100	8.770768
17	2	722.4	0.067188	74	4.969606	516	0.036875	62	3.905394	995.4	0.064063	70	4.578026
18	2	1060.8	0.099375	100	7.170348	1126.8	0.082813	100	6.987148	1712.4	0.11	100	7.120484
19	10	13775.4	1.539375	42	2.93594	13077.6	1.186875	14	1.004708	14733.6	1.198125	12	2.162133
20	3	1347	0.13	100	6.040049	2092.2	0.158438	100	5.879193	2154.6	0.14375	100	5.826021
21	3	1127.4	0.109688	100	6.136232	783.6	0.059688	98	5.958011	1748.4	0.117813	100	5.944329
22	10	14781	1.67	20	1.070084	12759.6	1.1575	6	0.519414	12219.6	0.9925	68	3.183566
23	2	1194	0.114063	100	5.169643	985.8	0.072813	88	4.667759	1893.6	0.123438	98	5.08816
	sum	**67515.6**	**7.214375**	**1992**	**123.1088**	**65715.6**	**5.603438**	**1848**	**114.2333**	**77796**	**5.810313**	**2060**	**126.173**

Table 5.7, Results on 23 benchmark problems with different dimensionality and complexities for *TDE*, *DE adapt* and *SACMPDE* variants.

		TDE NP=30, CR=0.5, F=0.5, MT=0.1				DE_adapt NP=30, CR=0.5, Fmin=0.1				SACPMDE NP=30, Fmin=0.1, Fmax=1, CRmin=0.05 CRmax=0.8			
f	n	fe	cpu	sr	Lambda	fe	cpu	sr	lambda	fe	cpu	sr	lambda
1	2	1384.8	0.114375	98	5.848955	1226	0.0966	100	5.9509	1361	0.1844	100	5.9485
2	2	1836	0.146875	98	6.498679	3089	0.2375	100	6.4184	2539	0.345	98	6.1783
3	2	1035.6	0.085625	100	5.299111	1031	0.08	100	5.3002	1105	0.1531	100	5.2955
4	2	938.4	0.075	100	5.773841	977	0.0744	100	5.8253	1068	0.1403	100	5.7495
5	4	2256	0.193125	100	7.128992	1765	0.1447	74	6.512	2569	0.3572	100	7.1365
6	10	14382	1.420625	26	2.050743	15000	1.4091	0	0.4123	13540	2.0612	88	4.1218
7	2	2058.6	0.165313	62	4.954935	3614	0.2781	82	5.7085	2732	0.3713	72	5.5357
8	10	2130.6	0.204688	100	4.746301	2190	0.2013	52	4.0725	3560	0.5403	100	4.7625
9	2	972	0.078438	100	3.614447	1101	0.0844	100	3.6152	1047	0.1359	100	3.613
10	2	1685.4	0.135625	100	6.871412	1744	0.1334	100	6.7151	1607	0.2091	100	7.5212
11	2	1376.4	0.110938	100	6.703105	2152	0.1656	100	6.5832	1492	0.1959	100	7.1386
12	2	1161.6	0.094688	100	6.473382	1874	0.1444	100	6.4855	1339	0.1756	100	7.2369
13	2	1033.8	0.084063	100	6.613696	1640	0.1259	100	6.5271	1213	0.1588	100	6.9358
14	2	1403.4	0.11375	100	4.567716	1396	0.1084	100	4.5719	1433	0.1888	100	4.5628
15	4	1536.6	0.131563	100	5.231399	1691	0.1375	100	4.9851	1918	0.2641	100	5.3547
16	2	1914	0.155313	100	8.771095	1903	0.1469	94	8.6791	1975	0.2587	100	8.769
17	2	671.4	0.053125	54	3.574668	839	0.0641	60	3.9094	732	0.0997	58	4.089
18	2	1344	0.1075	100	7.202152	1682	0.1291	98	7.0303	1450	0.1909	100	7.2894
19	10	10224	1.008438	72	4.041113	14984	1.405	2	1.1657	14904	2.2556	4	1.9948
20	3	1682.4	0.138438	100	5.883125	2911	0.2306	100	5.8251	1769	0.2375	100	6.2047
21	3	1362	0.113438	100	5.885658	2362	0.19	100	5.8187	1434	0.1941	100	5.9677
22	10	14151	1.397188	32	1.87396	15000	1.4194	0	0	10332	1.5478	100	4.7965
23	2	1657.2	0.135313	98	5.086657	1711	0.1319	100	5.1782	1503	0.2013	100	5.1675
	sum	**68197.2**	**6.263438**	**2040**	**124.6951**	**81882**	**7.1383**	**1862**	**117.2897**	**72622**	**10.4666**	**2120**	**131.3699**

Table 5.8, Results on 23 benchmark problems with different dimensionality and complexities for *SACPDE* and *SDE* variants.

		SACPDE NP=30, Fmin=0.1, Fmax=1, Fc=0.1, CRc=0.1				SDE NP=30, OPmin=0, OPstd=0.7			
f	n	fe	Cpu	sr	lambda	fe	cpu	sr	lambda
1	2	1396	0.0925	100	5.9546	1258.2	0.0828	100	5.9561
2	2	2188	0.1366	100	7.8582	1623.6	0.1009	100	6.4542
3	2	1157	0.0734	100	5.299	999	0.0634	100	5.2985
4	2	1026	0.0653	100	5.7786	939	0.0581	100	5.8118
5	4	2477	0.1694	100	7.1293	2057.4	0.1397	100	7.1152
6	10	11678	0.9178	100	4.2783	6154.8	0.4822	22	1.5407
7	2	2641	0.1663	58	5.4513	1666.2	0.1069	54	4.563
8	10	3022	0.2356	100	4.8433	2486.4	0.2003	100	4.6056
9	2	1066	0.0669	100	3.6137	963.6	0.0591	100	3.615
10	2	1740	0.1075	100	7.0864	1442.4	0.0909	100	6.9766
11	2	1472	0.0916	100	6.6666	1357.2	0.0956	100	6.6157
12	2	1397	0.0875	100	7.1548	1180.8	0.0747	100	6.8184
13	2	1166	0.0737	100	6.7965	1025.4	0.0638	100	6.7162
14	2	1607	0.1016	100	4.5645	1305	0.0831	100	4.5702
15	4	1820	0.1222	100	5.457	1558.8	0.1053	100	5.1084
16	2	2117	0.1344	100	8.7699	1905.6	0.1256	100	8.7707
17	2	918	0.0575	70	4.836	813.6	0.0575	70	4.6386
18	2	1601	0.1006	100	7.3061	1293.6	0.0869	100	7.078
19	10	14464	1.1369	52	4.2095	7655.4	0.5975	16	2.0807
20	3	1862	0.1203	100	6.0453	1674.6	0.1075	100	5.9487
21	3	1528	0.1013	100	6.1109	1335.6	0.0912	100	5.9552
22	10	10198	0.8153	100	4.8979	5664	0.4441	30	1.3969
23	2	1829	0.1159	100	5.1681	1544.4	0.1009	98	5.0886
	sum	**70370**	**5.0901**	**2180**	**135.2758**	**47904.6**	**3.418**	**1990**	**122.723**

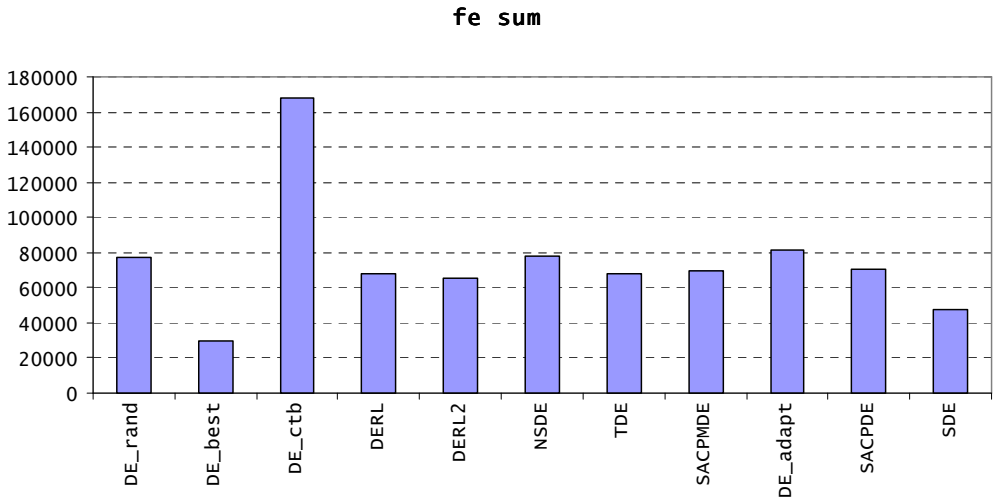


Figure 5.24, Sum of the function evaluations for the eleven DE variants over 23 SO problems.

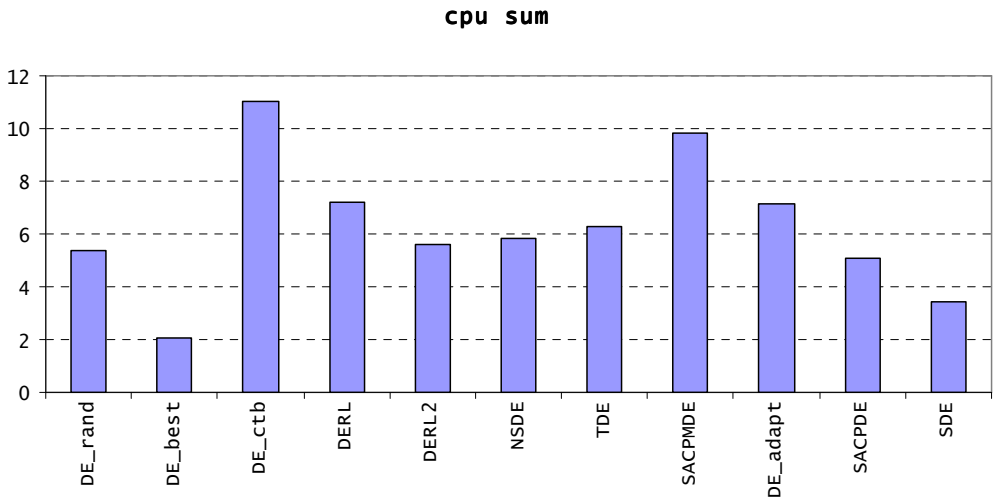


Figure 5.25, Sum of the cputime for the eleven DE variants over 23 SO problems.

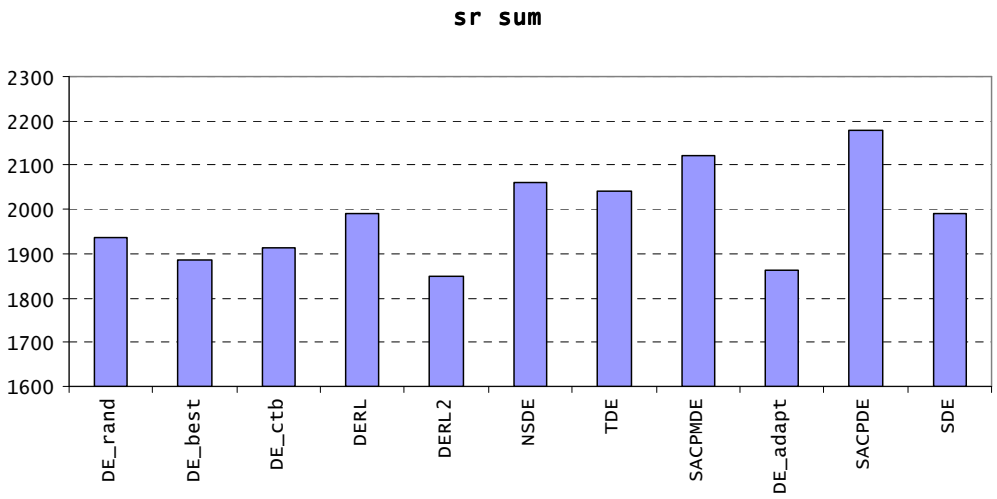


Figure 5.26, Sum of the success rates for the eleven DE variants over 23 SO problems.

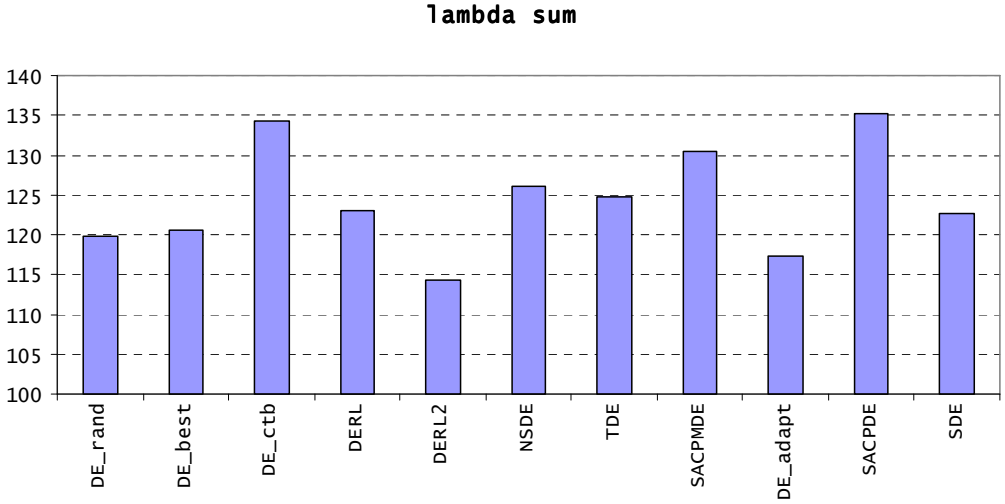


Figure 5.27, Sum of the lambda achieved for the eleven DE variants over 23 SO problems.

As clear from Figure 14, the number of Σfe moves approximately around the value 75000 for most of the variants; *DE best*, *DE current-to-best* and *SDE* are exceptions. This behaviour reflects indirectly the choice of unique *CR*=0.5 for the whole basic variants, which implies a common convergence speed till the loss of diversity inside the population, since this parameter seems to be the most significant.

However, fe is not meaningful alone: the most important is the sr of the algorithms (Figure 16); analyzing the seven variants without adaptive schemes (from *DE random* till *TDE*), the best algorithms are *NSDE* and *TDE*: they show high sr, good accuracy and acceptable cpu (Figures 16, 17 and 15). Notice that *NSDE* has the same structure as *DE random* with a randomization of scaling

factor *F* that allows different step length on perturbation, while *TDE* uses on a small fraction of population the fitness feedback in order to direct the perturbation; even if *TDE* spends less fe than *NSDE* to converge, the fitness feedback evaluation takes some cpu time; this behaviour is verified when the mutation strategy adopts further sophistications. Third for feats in this seven is *DERL*, with a good sr and lambda, faster convergence speed in terms of fe but with more cpu time consumed: in fact it uses in every mutation fitness feedback, increasing the computation time. *DERL 2* has poor performances compared with the others algorithms; it has low accuracy and the worst sr. *DE best* reflects its greediness in fe and cpu, the smallest in the test, compensated by less sr and lambda practically equal to *DE random*; as expected, this practice is faster but it should be used only in not complex problems: this condition is not often present; the risk is to lost the global optimum. The last basic variant of the seven is *DE current-to-best*: it has the lowest convergence speed, spending more than 160000 fe for all the test, the double than the others. On the other hand the accuracy is the highest, but the sr is unsatisfactory compared with *NSDE*, *TDE* and *DERL*. The classic version, *DE random,* is collocated on the average for performances, with more success than *DE best*, *DE current-to-best* and *DERL 2*, with good value for cpu and acceptable lambda. Its lack is on the setting for complex problems; this lack could be overcome using the improved version like *NSDE*, *TDE* and *DERL*, or an adaptive/self-adaptive algorithm.

In fact, the best variant on the whole test is *SACPDE*, with the highest success, high lambda and high convergence speed in terms of cpu and fe comparable with the others. Also *SACPMDE*, the modification of *SACPDE*, has good performances, with the second sr of the test, good accuracy (`lambda`) but really high `cpu` time, especially because the `fe` is comparable with the other variants: this is due to the amount of fitness information necessary for its mutation procedures. Anyway it is a reliable version. *SDE* is an interesting self-adaptive possibility, since its high convergence speed both in `fe` and `cpu`, with good `sr` and sufficient accuracy. *DE adapt*, that uses only scaling factor adaption rule, reflects the low sensitivity of F in this test: lambda and `sr` are insufficient, no improvements respect *DE random* in terms of `fe` and bad `cpu` are achieved. It is important to notice the setting of these adaptive schemes has low sensitivity on `sr` and `lambda`, as reported Table 8 for *SACPDE*:

Table 5.9, Summed results of the measures of the optimization by *SACPDE* on 23 SO problems with two different settings.

SACPDE	**Σfe**	**Σcpu**	**Σsr**	**Σlambda**
Fc=0.1 CRc=0.1	70370	5.0901	2180	135.3
Fc=0.25 CRc=0.25	68244	4.9027	2176	131.3

The new setting, which enhances the *F* and *CR* evolution during the evolution of the population, increases slightly the convergence speed and the `sr`, diminishing a little the accuracy; since the range for these two values is between `0.05` and `0.3` and our test uses coherently the literature recommendation, the superiority of this technique is proved over its setting.

As last comparison for SO, *DERL* with appropriate *CR*=0.1 and a mixing between self-adaption of *SACPDE* and *NSDE* scaling factor selection are tested on the 23 benchmark problems, called *SACPDE-NS*. The results are reported in Table 9.

Table 5.10, Summed results of the measures of the optimization performed by *DERL* and a mixed variant *SACPDE-NS* on 23 SO problems.

DERL	**Σfe**	**Σcpu**	**Σsr**	**Σlambda**
F=0.5 CR=0.1	71931	7.3756	2228	144.5
SACPDE-NS				
Fc=0.1 CRc=0.1 NS=0.5	64410	4.5964	2160	135.3832

The first is the most reliable result obtained by correct setting of the first seven variants (*NSDE* and *TDE* with low `CR` don't reach these performances) in terms of `sr` and `lambda`, with a little payment in convergence speed. This result is followed by *DE random* with *CR*=0.1 (see Table 3) and *SACPDE* (Tab 8). Of course this result is driven by tuning found by previous evaluations.

The second in Tab 9, called *SACPDE-NS* (*SACPDE* with Neighbourhood Search) has the *NS* scaling factor randomization (Gauss and Cauchy random variables) instead the provisional uniform randomization, which assures more generality. The improvement is in the accuracy achieved but especially in the convergence speed compared with *SACPDE* (`Fc=0.1, CRc=0.1, Fmin=0.1`).

5.1.2 Multi-objective optimization

This case study uses three benchmark problems, ZTD1, ZTD2 and ZTD3 proposed in [60] and reported in Appendix B, used often as comparison in literature between algorithms for multi-objective optimization. These three problems have high dimensionality (*n*=30) and a really restrict domain, defined between 0 and 1. Farther, the Pareto front is practically on the lower border of the domain, so, even the approach to the real solution is a difficult task for many EAs in multi-objective optimization.

The features of these problems are thought to demonstrate the exploring abilities and the accuracy achieved by the algorithms, even in an artificial complex scenario as these three benchmark problems.

In this case study the algorithms tested are *GA-toolbox*, *MOGA* and *MODE*.

The algorithms' setting

The *GA-toolbox* setting is similar to the setting already proposed in Section 5.1.1 for the single-objective optimization, with some difference in stopping criteria and some shrewdness necessary since the multi-objective nature of the problem:

The setting used is:

```
'PopInitRange'          [low;up]
'PopulationSize'        200
'EliteCount'            30
'CrossoverFraction'     0.7
'Generation'            500
'ParetoFraction'        1
```

The benchmark problems tested have 30 dimensions, so the initial ranges `low` and `up` are two arrays of 30 values each one. The lower bound is an array of zeros and the upper one is an array of ones. The population size is increased, since the difficulty of the problem, and the number `EliteCount` is set as the dimensionality of the problem. The crossover, ever single-site, has probability 0.7 and the maximum number of generations is 500. This is the only stopping criterion for the multi-objective optimization with this tool. The value `ParetoFraction` represents the fraction of the final archive respect to the population size. This value is one, since the desired number of non-dominated solutions is the same as the population size.

A default option for the multi-objective version of the tool is the distance measure applied to the solutions on the main archive: this option allows the removing of non-dominated solutions too close in terms of fitnesses if the archive is full. This practice should increase the diversity and the density of the Pareto front obtained, moving the exploration toward regions less dense of solutions.

The *MOGA* setting is similar to the previous one for *GA-toolbox*:

```
PopInitRange            [low;up]
PopulationSize          200
CrossoverFraction       0.7
Generation              500
ArchiveDim              200
```

EliteFraction	1/4
w_1	0.3
w_2	0.7

The elite fraction is the fraction of solutions from the archive used for the elitism concept. The archive dimension is set equal to the population size, in order to compare the results with the same number of non-dominated points.

Further, the algorithm has a weighting option for the multi-objective optimization: the objective function with higher weight has more attention on the optimization rather then the others. In our case the two weights are set toward the second objective function, more difficult to optimize for the three benchmark problems.

In multi-objective options, *MODE* uses three of the eleven variants allowable for the single-objective optimization. These three variants work also for multi-objective options, since they do not require fitness feedback information. The *MODE-III* version implemented into this tool does not use ranking during the evolution, so the superiority concept cannot be used for this situation. The selection of chromosomes for reproduction could be made only choosing randomly from the population, feature present only in three variants:

- *DE random*
- *NSDE*
- *SACPDE*

The population size and the number of maximum generations allowable are the same as for *GA-toolbox* and *MOGA*. The stopping criterion involves only the maximum generations, as for the others.

Population size NP	30
MAXGEN	500

The basic setting for the parameters of any variant is the same as for single-objective optimization presented in Section 5.1.1:

1. *DE random* F=0.5, CR=0.5
2. *NSDE* CR=0.5, NS=0.5
3. *SACPDE* Fmin=0.1, Fmax=1, Fc=0.1, CRc=0.1

Some sensitivity, as for the single-objective optimization test, are tried on *DE random* and *NSDE*. Also for the multi-objective optimization the

parameters play an important role in the convergence speed and accuracy, especially for these complex multi-objective problems.

SACPDE does not need sensitivities, since in the single-objective optimization its behaviour is practically independent on the parameters (see Table 5.8).

Measures

Two intuitive measures for the Pareto front are the computation time and the number of non-dominated solutions in the last archive. For the GAs algorithms the number of solutions is a parameter of the optimization; for *MODE* this value becomes significant, since it represents the goodness of the mutation technique used. The higher this value (the maximum is the population size), the higher is the ability of the variant to attain the Pareto front.

In order to compare the Pareto fronts obtained by the algorithms, a direct comparison between solutions is performed as in [54].

Figure 5.11 shows schematically the concept of the comparison: two Pareto fronts are obtained (red-circle and blue-square) by two algorithms; the black-solid line is the overlap of the fronts and in this case no front dominates the other. Anyway, a fraction of the frontier marked with circles dominates the square one on the left side of the plot, while a fraction of the latter dominates the first one (right side of the plot). So, for anyone frontiers, one can determine the fraction of the dominated solutions and the fraction of the dominant solutions with respect to the other frontier.

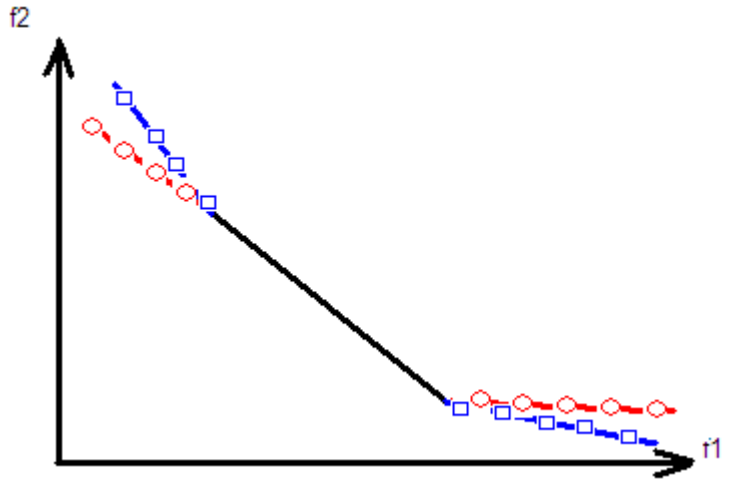


Figure 5.28, An example of two Pareto front achieved. The black-solid line is the overlap of the two frontiers. The solutions marked with circle dominate the square one only on the left side of the graph, whereas the latter dominates the first on the right.

Results and sensitivities

As first comparison we report the multi-objective optimization results on the benchmark problem ZTD1 (see Appendix B) for *GA-toolbox*, *MOGA* and *MODE* with the implementation of *DE random*. Figure 18 plots the Pareto front obtained with the three algorithms.

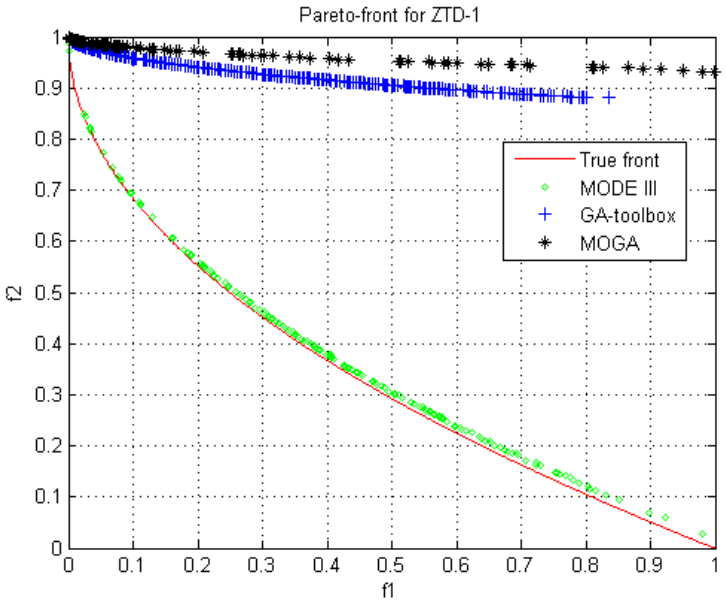


Figure 5.29, MO performed on ZTD1 benchmark problem with *MODE*, *GA-toolbox* and *MOGA*.

The inability of the two GAs are clear. No solutions from their final archive reach the true Pareto front. The complexity of the problem does not allow to the two GAs neither a significant approach to the true Pareto front. The algorithm *MODE* with the *DE random* implementation not tuned is very close to the front.

The direct comparison is not applied in this case, since there is no need to evaluate numerically the goodness of DE.

The optimization is then performed on ZTD2 problem (Appendix B). Figure 19 plots the solutions found by the three algorithms. *MODE* is another time run with *DE random* variant.

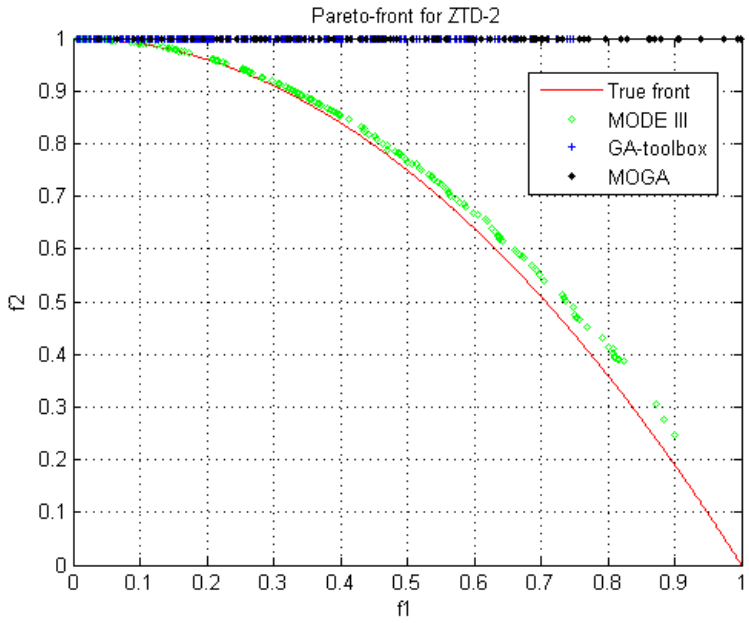


Figure 5.30, MO performed on ZTD1 benchmark problem with *MODE*, *GA-toolbox* and *MOGA*.

Also this time the two GAs fail completely in the search of the Pareto front. Also for this test the attainment of *MODE* is much better, but the Pareto front is not reached.

Neither here the direct comparison is used.

These two problems, in fact, are really complex, since the high dimensionality, and the Pareto front lays on the boundary. After that, the nature of the problem forces the usage of real-encoding: DE is properly thought for these situations, while GAs work much better on quantized problems. The superiority of DE, at least in these situations, is undoubted. Some significant help is given by the specific repair rule adopted. The bouncing-back approach for satisfaction of the boundaries is particularly well-chosen for these problems.

ZTD3 is more complex than the previous two ZTD1 and ZTD2, and then the two GAs, *GA-toolbox* and *MOGA*, are not used in the comparison for this benchmark problem.

MODE in this complex situation is highly helped by the shrewdness of the bouncing repair rule, since the Pareto fronts for ZTD1, ZTD2 and ZTD3 are on the boundary. Anyway, the two GAs fail completely the search also for the two simplest problems ZTD1 and ZTD2 respect to ZTD3.

A comparison is made between the three *MODE* variants: *DE random*, *NSDE* and *SACPDE*. Table 5.10 reports the searching times and the number of non-dominated solutions found in the last population for these three variants on the three problems tested: ZTD1, ZTD2 and ZTD3. Remember that *MODE* does not

use an archive for non-dominated solutions, but it carries on the entire population to the Pareto front: at the end, the dominated solutions are removed, skimming the population. For this reason only with high number of generations the entire population should achieves the front.

Table 5.11, Cputimes and number of non-dominated solutions found by the three algorithms in the three tests ZTD1, ZTD2 and ZTD3 at the end of the searchs. The initial population number of *MODE* is NP=200.

		DE random	***NSDE***	***SACPDE***
ZTD1	Cpu	6.72 s	7.28 s	6.76 s
	N	175	124	194
ZTD2	Cpu	6.59 s	7.25 s	6.85 s
	N	178	118	195
ZTD3	Cpu	6.61 s	7.54 s	6.78 s
	N	147	96	167

Figures 20, 21 and 22 plot the solutions of the three *MODE* variants for the three test problems.

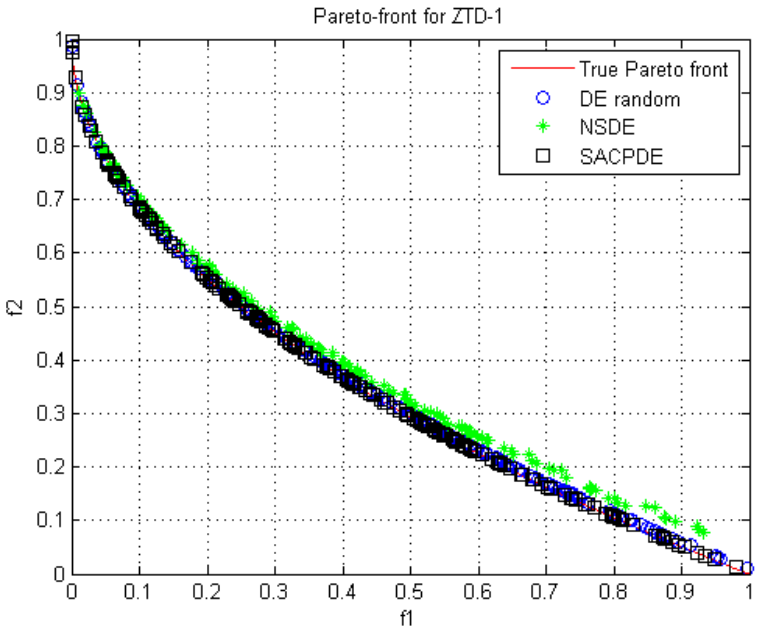


Figure 5.31, *DE random*, *NSDE* and *SACPDE* Pareto fronts obtained in MO for ZTD1.

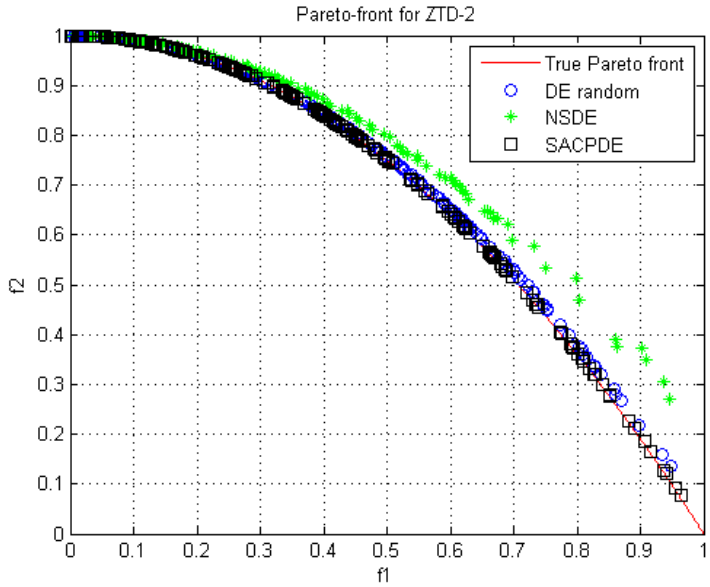


Figure 5.32, *DE random*, *NSDE* and *SACPDE* Pareto fronts obtained in MO for ZTD1.

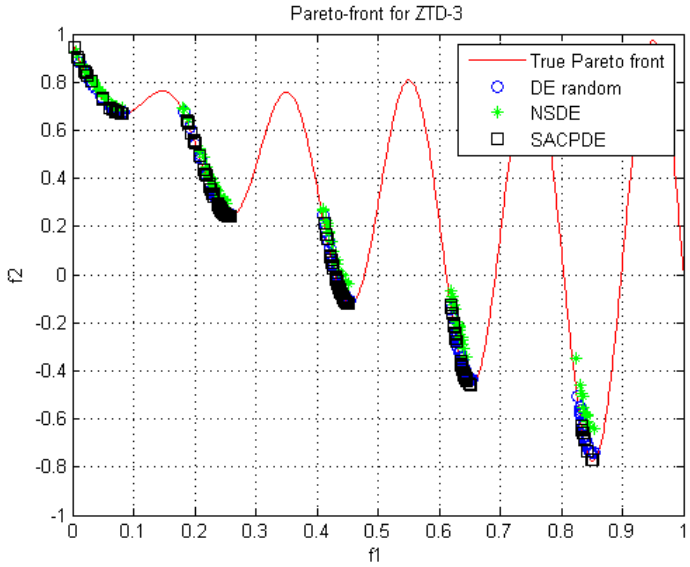


Figure 5.33, *DE random*, *NSDE* and *SACPDE* Pareto fronts obtained in MO for ZTD1.

All the three variants have a good approaching to the Pareto fronts respect to GAs, but *SACPDE* outperforms the other two variants, reaching the Pareto front with more accuracy; a direct comparison is made between the three fronts obtained. So, a 3x3 table is reported: the variant present at the beginning of the row is compared with the variant on the first cell of the column. Then, for each intersection between rows and columns, two cells are present: the first contains

the percentage of solutions of the row-variant that dominates the column-variant's ones, the second contains the percentage of solutions dominated by the column-variant.

From Table 5.11 the superiority of *SACPDE* respect to the other two variants on the ZTD1 is deducible: *SACPDE* outperforms *DE random* and *NSDE* with the 61.78% and the 96.34% respectively of its front respect the two variants. Furthermore, no points of *SACPDE* are dominated. Between the other two variants, *DE random* results to be better than *NSDE*, since no points of the latter dominate some solution of the first.

Table 5.12, Direct comparison between the three variant tested of *MODE* for ZTD1. The variant on the line is compared against the variant on the column. The first value in the intersection is the fraction dominant solutions of the row against the column, the second is the fraction of dominated solutions.

	DE random	***NSDE***	***SACPDE***
DE random	-	94.05%	0%
	-	0%	62.16%
NSDE	0%	-	0%
	90.08%	-	92.56%
SACPDE	61.78%	96.34%	-
	0%	0%	-

Farther, the number of final non-dominated solutions of *SACPDE* is sensibly greater compared with the other twos, and the cputimes are slightly greater than the *DE random*'s ones.

Apparently, the problem for *DE random* and *NSDE* is the parameters' setting.

Sensitivities are then reported: as experienced in single-objective optimization, the *CR* is the most significant parameter, especially for complex problems as these ones. Furthermore, *NSDE* has an interesting ability to explore the neighbourhood: manipulating the *NS* parameter, the search could be redirected on the neighbour, diminishing the probability to have high length for the jumps.

Then, *DE random* and *NSDE* are tested with *CR*=0.3. Table 12 reports the number of non-dominated points found in the last population for the two variants on the three problems and the percentages of dominant and dominated fraction of solutions over the other variant. The diminishing of *CR* increases the number of solutions in the last population, both for *DE random* and *NSDE*. As for single-objective optimization, the diminishing of this parameter is beneficent for complex problems.

Table 5.13, Number of final solutions in the last population and direct comparison between the two parameters' dependant variants of *MODE: DE random* and *NSDE*. The values are reported for the three benchmark problems.

	ZTD1		ZTD2		ZTD3	
	N	Comp.	N	Comp.	N	Comp.
DE random	199	65.66%	199	74.87%	169	64.5%
		0%		0%		0%
NSDE	178	0%	174	0%	120	0%
		60.11%		73.56%		64.17%

Now, with low *CR*, *DE random* becomes better than *SACPDE*, as reported in Table 5.13 for ZTD1. The percentage is small, but no solutions of *SACPDE* dominate solution achieved by *DE random*.

Table 5.14, Direct comparison between *DE random* with a tuned setting (*CR*=0.3) and the *SACPDE* variant. The values reported are referred to ZTD1.

	DE random	***SACPDE***
DE random	-	21.72%
	-	0%
SACPDE	0%	-
	19.90%	-

Nonetheless, the results with *NSDE* are still unsatisfactory respect to the other two variants: a tuning on *NS* is made, leaving *CR*=0.3 and imposing *NS*=0.8, so increasing the neighbour search ability of the algorithm in these problems with tight domain. A new direct comparison is then made with the results previously obtained for *DE random* with *F*=0.5 and *CR*=0.3.

Table 5.15, Number of final solutions in the last population and direct comparison between the two parameters' dependant variants of *MODE* with an opportune setting*: DE random* (*F*=0.5, *CR*=0.3) and *NSDE* (*CR*=0.3, *NS*=0.8). The values are reported for the three benchmark problems.

	ZTD1		ZTD2		ZTD3	
	N	Comp.	N	Comp.	N	Comp.
DE random	199	27.14%	199	29.30%	169	49.7%
		0%		0%		0%
NSDE	191	0%	194	0%	162	0%
		26.65%		26.80%		43.83%

Neither this time *NSDE* outperforms *DE random*, but the numbers of final solutions on the three tests are now comparable and the percentages of dominated solutions are sensibly decreased. Only for ZTD3 these percentages remain high.

5.1.3 Conclusions

Concluding, the usage of adaptive/self-adaptive algorithms is recommended for untrained users, both in single-objective and multi-objective optimization, since the easy setting and their good abilities in most cases, due to the intrinsic flexibility. They use the basic recombination of *DE random*, reaching anyway high performances, symptom of the sensitivity of this powerful recombination on the setting parameter: mixing reproduction greediness with adaptive scheme doesn't give the expected improvements, since the greediness is introduces just to overcome the lack leaved by the specific setting.

SACPDE is the most attractive variants proposed in the tool *MODE*, since its intrinsic flexibility and ease on the setting. Anyway, the performances of this variant could be outperformed by other variant opportunely tuned.

However, the usage of modifications in reproduction phase is reserved to expert users, especially for high dimensionality, where their performances could reach the adaptive ones, since the difficulty of the setting. Greedy sophistications improve the performances but needs unavoidable further information and knowledge on the strategy characteristics and behaviours. The *CR* parameter is the most influent, especially for complex functions, and low values ($CR<0.3$) assure reliability of the optimization, while high values improve the convergence speed, impoverishing the accuracy and the success. The population size doesn't play any meaningful role and *F* has low impact on the results in most cases. The common value is 0.5, but a uniform randomization $F\sim U(0,1)$ could be done without any strong implication.

5.2 A real case study. Giant oil field integrated production asset: a highly constrained optimization for productivity

This section presents the optimization on a real case of an integrated asset for production of hydrocarbons from a giant oil field composed by a gathering system coupled with a process plant.

The optimization, performed by a tool, equipped with Differential Evolution, specifically developed inside the ENI E&P Prod division, takes into account a production line of the entire system with a difficult management, since the presence of constraints both in the gathering system and the process plant. The integration between the two environments of the same asset represents one of the most common difficult tasks inside the oil companies' production management.

5.2.1 Introduction

The optimization is a difficult task for real cases, especially when the number of variables and constraints is high and the interaction between them is not completely clear. The uncertainties in real cases are often significant, compromising the analytical search of the optimum. Several instruments are offered to the industries, like simulation programs as more reliable, which increase the understanding of the models built to represent the reality. Anyway, the correct variable's setting for a model is often far from the mind's ability to obtain the optimum. Evolutionary algorithms represent a good choice, especially when classic optimization methods fail, because of the high complexity of the model or the presence of severe conditions.

Differential Evolution, a particular type of evolutionary algorithm, has been used to build an optimization tool for oil industry management. This algorithm is chosen since its reliability and rapidity to find the global optimum.

The Section 5.2.2 explains the general description of the real cases within this tool could be utilized, whereas Section 5.2.3 depicts the real case under investigation. Section 5.2.4 gives an overview of the tool properties while section 5.2.5 goes deep into the algorithm's strategies and shrewdness. Section 5.2.6 collects and explains the results.

5.2.2 Problem's Generalities

The task of this optimization is to find the correct setting of a specific integrated asset of a giant oil field in order to enhance the oil production. The production system under the Prod division administration and within it could work is composed by two different environments:

1. the gathering system, that starts from the wellhead till the separators' collectors;
2. the process plant, that takes the product fluids and processes them in order to reach the required specifications.

These two environments represent the typical production chain managed by the production division of an oil company. Figure 5.19 shows the entire production system for an oil field.

Gathering system

The gathering system is a complex scenario within the number of valves, pipelines, pumps, compressors, separators, test separators and collectors are joined in order to create a sufficiently flexible and operational network to carry the reservoir fluids to a storage or processing area. Actually a complete production system consists also of a reservoir and a well (see Figure 5.19), connected with equipment at the top of the producing wellhead, called "Christmas tree", used to control the flow. The gathering system starts from

here till the final separators or collectors and comprises all the other wells connected to the network.

As known, the driving force for an oil or gas production system is usually the pressure present in the reservoir: the pressure difference (in Figure 5.19 the pressure losses have an arrow from lower to higher pressure, while the flow has inversed direction) drives the reservoir fluids into the wellbore, and from here through the tubing till the wellhead. The Christmas Tree is equipped with safety and control valves; after it usually a surface choke valve is present: this equipment is used to control the flow rate. The inlet choke is the first valve before the network: form here till the final collector pipelines are installed in order to transport the fluid. Inside this portion of gathering system, pumps or compressors could be installed, if driving force is necessary.

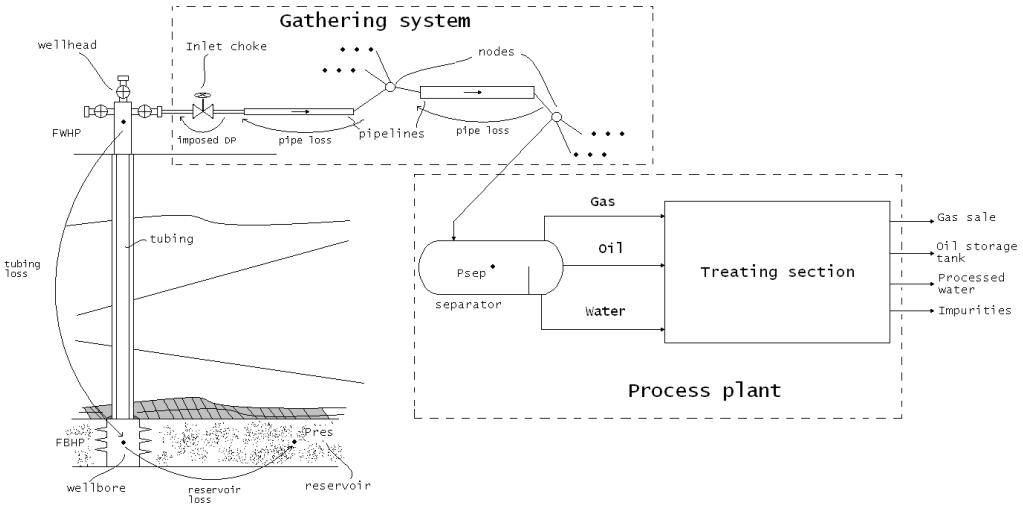


Figure 5.34, The production chain for a hydrocarbon field.

The inlet choke valve is the control equipment of the production system: adjusting the choke size it's possible controlling the flow. For example, a closing of the choke valve causes a back-pressure in the network, increasing the flowing bottom-hole pressure FBHP (the pressure at the beginning of the riser tubing), diminishing the driving force for the reservoir fluid; since the reservoir pressure is constant, the higher the FBHP, the lower the pressure drop between reservoir and wellbore and the lower the flow rate. The inlet choke doesn't have a completely direct control on the flow rate, since it is connected to a network: a choke opening on another valve could change the network pressure in some nodes, reflecting this pressure change upstream on the other lines or nodes. That means the pressure drop on a choke valve is not the unique variable that affects the flow rate from the reservoir, but the entire network layout and the other flow streams influence the productivity. This complex

scenario makes difficult a complete understanding of a production asset and its management.

A second control item in the gathering system is the end pressure of the network (usually the separator pressure). The higher this pressure, the lower is the potential driving force available and lower is the production. Anyway also in this case the behaviour is not directly proportional to the end pressure: a diminishing of it increases the production but at the same time also the pipeline pressure losses increase.

Nevertheless, the choice of this end pressure is not a banal task: even if it seems counter-productive, this pressure value is usually high. The reasons are several: the main one is the multi-phase nature of the reservoir fluids. These fluids commonly are a mixture of hydrocarbon fluids and dissolved gases, water, and non-hydrocarbon gases like H_2S, CO_2 and N_2. The amount of dissolved gas is a pressure dependant variable; hence, with the diminishing of the pressure the amount of free gas increases, altering the flow rate in the pipelines. Moreover, the gas is usually a secondary product in oil field production assets, since its lower economic value. Other relevant reasons depend on the process plant layout.

Process plant

When the main product is oil, as in our case study, at the end of the gathering system a process plant is installed. The main purpose of a process plant is to treat the reservoir fluids in order to reach the technical specifications for the sale or storage. Since the multi-phase nature of the fluids, several products could be produced and processed. Anyway, also the gas fields need a process plant in order to treat, clean and sweeten the gas from sour gases like CO_2 and H_2S. The end product specifications may be defined by a customer, by transport requirements, or by storage considerations. Table 5.15 shows typical specifications and conditions for sale.

Table 5.16, Typical specifications for a process plant released fluids.

	True Vapour Pressure TVP	<83 kPa @15°C
	Base Sediment and Water BS&W	<0.5% vol
	Temperature	>Pour point
	Salinity (NaCl)	<70 g/m^3
	Hydrogen Sulphide (H_2S)	<70 g/m^3
Gas	Liquid content	<100 mg/m^3
	Water dew point at -5°C	<7 Pa
	Lower Heating Value LHV	> 25 MJ/m^3
	Composition CO_2, N_2 and H_2S	National spec.
	Delivery pressure and temperature	Transport spec.
	Wobbe index	<52 MJ/m^3

A basic process plant has one, two or three separators as conjunction between gathering system and plant operation units. The separators could be two or three-phase: the first type splits gas from the mixture oil-water, the second typology divides the three phases into three distinct lines: gas, oil and water. The number of separation stages is a design variable of a process plant. A two stage-separation is the common disposition. The reason is economic: one stage has low installation costs but low efficiency in oil-recovery; increasing the stage number, the oil production increases but also the initial costs increase. Three stages usually are not justified, so the two-stage is the typical layout.

After the separation unit the gases released are compressed (the second separation is performed at lower pressure than the first) and sent to the gas treating section. The units of the gas treating section are the acid gas removal unit, the dehydration unit and the condensate recovery unit. The first has to remove CO_2 and H_2S, using usually solvents, the second one has to remove the water present in the gases in order to prevent the formation of a free water phase and to inhibiting the hydrate formation, and the third has to recover the heavy components from the gas that could condensate in the transportation phase. This last unit adjusts the Wobbe index of the sale gas.

On the other hand the separated oil is sent to the stabilizer (together with the removed gasoline) in order to remove completely the light gases like methane and ethane dissolved in the liquid phase: the reason is that they have high tendency to flash in the storage tank, decreasing quickly the partial pressure of the other gases dissolved. This rapid change in partial pressures increases also their tendency to flash to vapours, decreasing the quality of produced oil. The aim of the stabilization process is to increase the amount of intermediate (C_3 to C_5) and heavy (C_{6+}) components in the liquid phase by a quasi-complete methane and ethane removing. The oil stabilized is then cooled and sent to the storage area. The TVP specification is reached managing the stabilizer operation conditions.

Figure 2 shows a classic scheme for an oil process plant.

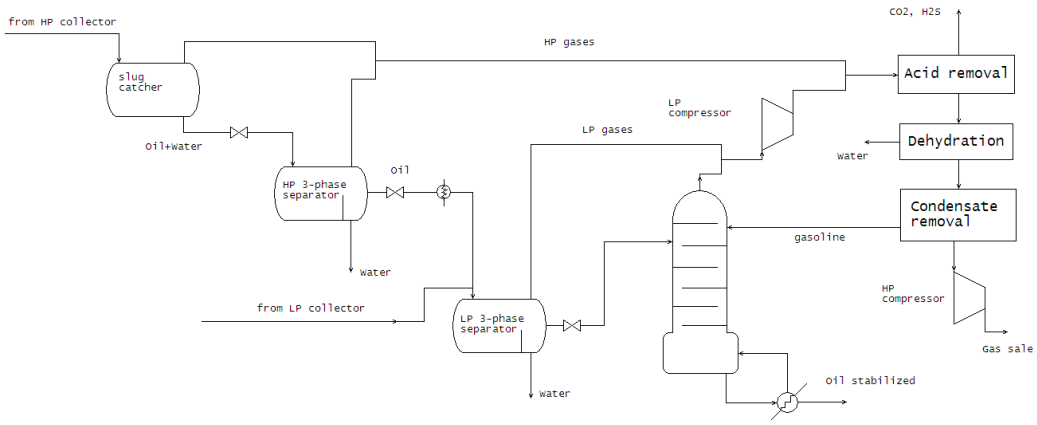


Figure 5.35, Simplified scheme for an oil process plant.

Several considerations could be done in a process plant. The design and operation of it is not easy, even if the plant has typically the same classic layout as previously described. The main difference may come from the fluid composition, the amount of gases dissolved, the sourness of them, the water and sulphur content and so on, that influence the plant management.

5.2.3 The case study

The asset under investigation has the previously described properties and layout. It is a specific line of a great giant oil field. The objective of this optimization is to enhance the oil production for this line, since the amount of released gas does not affect the economics of the field as the oil.

The line has 8 productive wells, divided in different clusters: the network has two high pressure branches connected to three and four well for each cluster (see Figure 5.21). The last one is separated and connected to the low pressure network. This scenario is complicated by the zone’s orography, because the wells are drilled on a higher level respect to the process plant, and the pipelines have latch and consequently complex multi-phase flow behaviour; the resulting pressure losses are functions of the liquid content on the pipelines.

The process plant has two separation stages; the high and low pressure networks are connected to these two item, so the high pressure separator takes the highest fraction of oil from the network; the low pressure separator takes the oil coming from the first separator and the oil coming from the low collector (Figures 5.20 and 5.21). In fact, in the plant there are three separators: one is a two-phase type and two are three-phase separators; the two-phase separator is used as slug catcher and it is posed at the beginning of the plant, connected to the high pressure collector. Between the two-phase and the first three-phase separator there are only 1-2 bar of pressure difference: practically they represent together the first separation stage. Each well has its Inlet choke

valve. The process plant has one stabilizer, divided into two sections coupled in the same column, one acid gas removal column, feed by MDEA regenerated in another column, a dehydration unit with glycol and a condensate recovery. The gases are compressed two times: the first from the separation and stabilizer pressure till 30 bar approximately and then sent to the treating section; the second after the condensate removal till the transport line pressure required, 70 bar.

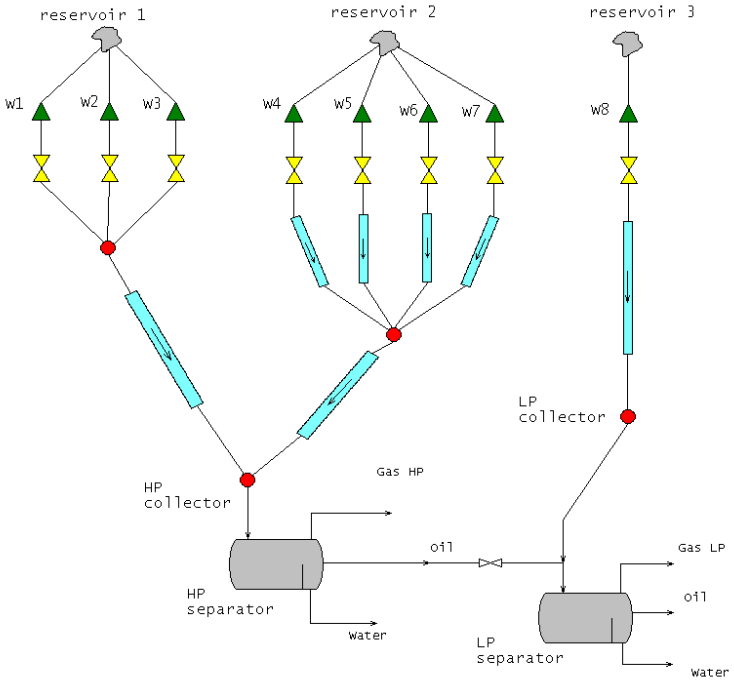


Figure 5.36, The gathering system for the real case study.

Each well has a different fluid composition and consequently different behaviour. The amount of gas present in the fluid is represented by the GOR [Sm^3/Sm^3] (Gas-Oil-Ratio), defined as the free gas flow over the oil flow at standard conditions. This number is significant in the production management, because a high amount of gas released in the network could cause unstable flows.

Another significant characteristic is the amount of acid gases present in the fluid: CO_2 and H_2S could provoke corrosion in the pipelines, even ungovernable, and make necessary an efficient acid removal unit in the process plant in order to obtain the sale and safety specifications.

The problem of this asset is the coupling of the fluid characterizations with the process plant. The initial design was made using different GORs and compositions from the actual ones, since work-over, acid cleaning and

recompletions are made in some well. Table 5.16 shows the critical properties of the wells' fluids.

Table 5.17, GOR and sour gas content of the reservoir fluids.

	GOR	H_2S	CO_2
	[Sm3/Sm3]	%vol	%vol
W1	516	2	24.3
W2	428	2.77	9.65
W3	638	1.21	38.9
W4	250	0.2	3.6
W5	155	0.15	3
W6	150	0.16	3
W7	117	0.15	2.66
W8	160	0.1	3.1

The most significant problem of the plant is the acid gas sent to the acid removal unit: since the wells from reservoir 1 (W1, W2 and W3) have high fractions of sour gases and high GOR values, their productions limit the capability of the plant. Producing from reservoir 1, the amount of gas incoming in the plant increases; besides, this gas is really acid and exceeds the maximum design of the treating section: the amount of CO_2 in particular could degrades the solvent action in the unit. For this reason, the cluster of reservoir 1 has reduced production compared with its potential production, and the cluster of reservoir 2, with lower GOR and sour gases content, covers a high fraction of available production.

Anyway, the almost closing of the reservoir 1 cluster is not the optimal solution, since some profit margin exists: a slightly different plant parameters setting and a controlled network management could give some advantage. Besides economic considerations, a decreasing of the well's production on the cluster of reservoir 2 elongates its production life. In fact, some constraints are present also for the well production: the bottom-hole-pressure FBHP must to stay over a specified value related to the reservoir. This value is usually the 70% of the static reservoir pressure. This constraint reduces the possibility of a premature well-dead and reduces the sand production into the wellbore, reducing the fouling in the separators.

Speaking about the reservoir 3, the oil production is relatively small, approximately less than 10% of the total oil production; furthermore, the GOR is smaller with respect to the wells of cluster 1, making its regulation irrelevant in the optimization.

Concluding, in this situation the plant seems to be the constraint of the asset, because of its limitations. The controls of this asset are the inlet choke valves,

the separator pressures, and some degree of freedom in the process plant, like the pressure difference between the slug-catcher and the first 3-phase separator, the conditions of the stabilizer column (head pressure and reboiler temperature) and some outlet temperature of heat exchangers. The treating section, since its complexity, is leaved untouched by this integrated optimization and managed in a second time.

5.2.4 Integrated optimization

The two environments are modelled by two different programs, which calculate with high accuracy the results of a specific setting in each environment.

The necessity of a tool which integrates the two environments is driven by the difficulty to have a complete view of the entire asset. In fact, it is simulate by two programs. Each simulation program has an internal optimization tool, but separated with the constraints of the other environment; in particular, the optimization of the production from a network point of view cannot include problems of the treating section of the process plant but it could be influenced only by general considerations (e.g.: maximum gas rate, maximum H_2S). On the other hand, the constraints' satisfaction in the process plant is highly dependant on the input hydrocarbon mass flow rate, given by the network environment. It is clear the tight interaction between these two systems: the Differential Evolution gives some advantage in this optimization, since it's an external optimization tool that finds the variables setting using the evolution strategy of the survival of the fittest, allowing at the same time a flexible constraints setting.

Anyway, this tool could also work only on the process plant environment. This option does not take the gathering system as part of the system. The algorithm then works only on the HYSYS variables.

Gathering system simulation

The gathering system is modelled by a Petroleum Expert product called GAP, from the IPM suite, commonly used in the Eni divisions. In this program each well needs complex specifications for the reservoir fluid extracted, the performance, layout and length of the perforated well and the length of the tubing: the combination of these characteristics defines the well production performance to the surface controlled by the choke valve regulation. In order to solve the network an end pressure point, the separator pressure, must be set. Running the simulation, the programs returns the oil, water and gas produced by each well, the pressures in each node and the pressure losses in the pipelines.

Process plant simulation

The process plant environment is modelled by an AspenTech product, HYSYS, equipped with several thermodynamic packages and several operation

units like separators, column, absorber, reboilers and heat-exchangers. The accuracy of this program is high, but the program management is not banal. This tool is one of the most spread simulation programs used in the oil and chemical industries. It has also logical controls and adjusting operators that give some automation to the simulation.

The integrated optimization tool

Since the necessity of accurate solutions, these two simulation programs are used in our evolutionary algorithm: the tool proposed uses Differential Evolution as basis for the optimization, and the two programs simulate each solution explored by the algorithm; practically they take as input the chromosome's variables, they simulate the solution proposed and returns the fitness value to the main algorithm: the implementation of this tool is only for single-objective optimization, since usually in oil industry the objective function to maximize is the profit of the asset. If many products are the output of the processing phase, the aggregate planning method is adopted in order to define a unique fitness function. The main algorithm is written in MATLAB, a Mathworks product. The choice of MATLAB is driven by the possibility of it to create a connection and an information crosstalk with the other simulation programs, which are equipped with coherent external interfaces.

The tool is then composed by the interaction of three programs:

1. MATLAB
2. GAP
3. HYSYS

The first plays a manager role of the variables, solutions and operations for the search of the optimum. It works as an automated operator, following the population based search of the evolutionary algorithms: it computes each population, it combines the solutions in order to obtain new perturbed solutions, it compares the children with the parents and then it selects the best in order to allow the evolution of the population. This process permits the attainment of the optimum of the integrated asset, since the two simulation programs are used as fitness evaluation: the manager MATLAB find, combining chromosomes, a set of variables, and it follows the production chain imposing in GAP its specific set of variables, running the simulation and taking the results of the gathering system as input for the process plant simulation. Then MATLAB sets the required variables in HYSYS and it runs the process simulation; the final result, in that case the oil production, together with the constraints of the two environments, are read and processed by MATLAB. This operation is repeated till a stopping criterion is met. Figure 4 depicts this cycle.

The structure of the tool is flexible, adaptable to many cases of the oil productivity management. An classic interface through spreadsheets of the

simulation programs allows an easy setting for variables and constraints with their boundaries.

Since the automation of this tool, any further human interaction with the simulations must be removed; the correct convergence of each solution is controlled by MATLAB, but the simulation files needs shrewdness like internal logical settings.

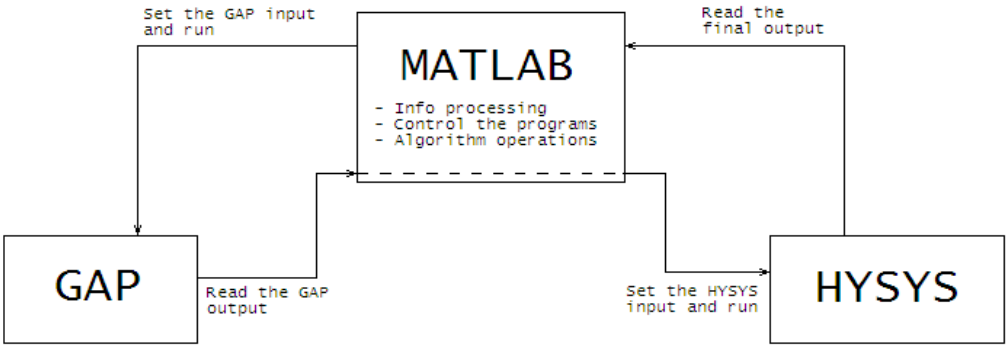


Figure 5.37, The interactions between the three programs: MATLAB, GAP and HYSYS.

Variables, constraints, results and interactions

This optimization comprises 14 variables together:

- 8 inlet choke imposed pressure losses
- 2 separator pressures
- Pressure difference between slug-catcher and first separator
- Stabilizer head pressure
- Stabilizer reboiler temperature
- Inlet temperature of the bottom stabilizer section

The two separator pressures are shared variables, since the separators are included in both environment and represent the conjunction. In this integrated optimization the separator pressures become variables, since in the separated one these values are a user defined conditions.

Some of these variables have defined boundaries:

Table 5.18, Variables' boundaries

	Lower	Upper	Unit
HP separator	20	40	[bar]
LP separator	12	18	[bar]
ΔP slug-sep	0.5	2	[bar]
Stab head P	7.5	9.5	[bar]
Stab reboiler T	160	190	[°C]
Stab bottom inlet T	65	120	[°C]

The other variables, the pressure losses through the choke valves, have only the lower boundary of 0, which means no control on the well, leaved completely opened. The upper boundary for this type of variable is not unique, since the pressure loss across the valve is not a direct control on the flow rate but it's influenced by the current network's pressure profile.

The constraints of the system, as previously anticipated, are both in the gathering system and in the process plant.

In the gathering system a required minimum value for each FBHP is related to the sand production and the life of the well: working over these values is recommended and not imposed. Anyway, for this optimization all the constraints should be satisfied. Table 5.18 reports these values, which are the 70% of the reservoir pressure layer from where they are producing.

Table 5.19, The minimum FBHP allowable

	FBHP min [bar]
W1	226.5
W2	226.5
W3	227.0
W4	217.6
W5	211.8
W6	221.0
W7	214.9
W8	209.4

The rest of the constraints are referred to the process plant. As specification design, the plant has maximum oil, gas and water capabilities defined as inlet values. After them, a significant constraint is the gas flow sent to the treating section, in particular the acid gas removal, expressed in actual volume flow, a pressure dependant property: since in the plant the low pressure compression unit carries the gases released from the low pressure separator and the stabilizer till the pressure of the high pressure separator, the setting of the latter influences deeply this constraint. The higher this pressure, the higher is the mass flow that passes through the treating section for a defined volume flow. Another constraint affected by the acid removal section is ratio between the CO_2 and H_2S fraction of these gases: over the specified value the unit doesn't work as required, since the high amount of CO_2 degrades the solvent action in the H_2S removal. Other two additional constraints could be the specifications of the Wobbe index and TVP: these depend respectively on the composition of the inlet fluids and the stabilized reboiler temperature.

Table 5.19 resumes the plant constraints.

Table 5.20, Constraints and specifications for the plant

	Value	Unit
Inlet oil [max]	37750	[bbl/day]
Inlet gas [max]	1450	[kSm^3/day]
Inlet water [max]	2200	[Sm^3/day]
Gas flow rate to the treating section [max]	42	[km^3/day]
CO_2/H_2S to the treating section[max]	21.4	[-]
Wobbe index [max]	52	[MJ/m^3]
TVP [max]	86	[kPa @100°F]

The fitness function of each solution is represented only by the oil produced by the plant and sent to the storage area, reported in standard conditions. No intermediate properties affect directly the objective function, the interaction is only indirect and internal to the system. The evolution should be able to altering the variables' setting in order to find the maximum oil production achievable that satisfies the whole constraints.

5.2.5 The algorithm strategies and properties

The Differential Evolution has many strategies adoptable and several constraint handling methods. The choice of implemented strategies is driven by the problem nature and the simulation times necessary to evaluate each solution.

The problem nature indicates that the optimization is mainly concentrated on the gathering system variables, since the oil production is high sensitive to the inlet choke opening; the process plant variables have marginal sensitivity on the fitness function; only in the constraints handling they have some weight, especially for the maximum Wobbe index and TVP. After that, the objective function with high probability does not have a complex shape but it could have some local optimum; this sentence is driven by the know-how in this type of optimization problems inside the Prod division. The main problem on the fitness function is the gas flow rate constraint, which excludes some portion of the domain: in fact, increasing the production, also the gas rate increases. This constraint affects sensibly the feasible area, especially in the domain of the well's cluster of reservoir 1, characterized by high GOR values (516, 428 and 638 Sm^3/Sm^3 for the three wells) and high CO_2 content.

The simulation times depends on the stability of a solution: if the solution is unstable both in GAP and HYSYS, the estimation-time for a solution could be 10 and 30 seconds respectively, with the possibility to waste approximately 40 seconds for a bad solution, discarded by the algorithm controlling the maximum error. These times are unacceptable, but continuing in the search, the number of

bad solutions diminishes drastically since they lie in an unstable region leaved soon unexplored by the evolution strategy.

Anyway, in order to obtain accurate solutions, 5 seconds for GAP and 7 seconds for HYSYS are considered necessary, so approximately 12 seconds for each chromosome is the expected time for each solution's evaluation. This feature influences the strategies adopted in the algorithm and the parameters' setting, especially for the population size and the stopping criteria.

Since very high optimization times are expected, the algorithm must be sufficiently faster and the same time reliable as possible.

DE strategies implemented

Given the previous features for the simulation of each solution and the presumed objective function's shape, the evolutionary algorithm implemented adopts three fast and reliable strategies already presented in Sections 4.1 and 4.2 and tested in Section 5.1:

1. *DE random*
2. *DE best*
3. *DERL*

These three reproduction methods are chosen chromosome by chromosome using some heuristic rule implemented in the program.

DE random is the basic mutation strategy originally proposed by Storn & Price (see Chapter 4).

DE best is the fastest algorithm since its greediness: this strategy speeds up the convergence when a clear direction is taken by the evolution. The implementation of a coherent heuristic rule allows avoiding misuse of its feature that could provoke premature stagnation to a local optimum.

DERL is a half-way strategy between *DE random* and *DE best*: it represents the correct mediation between greediness and randomness, their strengths.

The crossover procedure adopted is the binomial type, recommended in high dimensionality and highly constrained situations.

Parameters' setting

These three strategies need only three parameters:

- Population size `NP = 20`
- Crossover rate `CR = 0.5`
- Scaling factor `F = 0.5`

The population size affects the optimization time: since the simulation time for each solution is significant, a high `NP` implies an unacceptable wasting time. This value must be as lower as possible: the dimensionality of the problem is high, 14 variables, but only 10 of them could be really significant. `NP=20` is

considered a correct setting: the low sensitivity of this parameter in the final solution has been proved in Section 5.1.

The crossover rate influences the convergence speed: this tool must be reliable and at the same time as faster as possible. As proved in section 5.1, the lower the `CR` the higher the success rate, especially for complex problems: this high reliability is then counter-balanced by a slow convergence speed. Since the probable simple objective function shape, low values are not necessary and `0.5` is considered satisfactory for this situation.

The scaling factor selection is not critical, thanks to the heuristic rules implemented: a randomization around the user-defined value is performed by them.

The heuristic rules adopted

The heuristic rules adopted in this optimization uses fitness information processing in order to give flexibility to the evolution. Since adaptive rules need information about the population diversity, the standard deviation of the fitness functions of the previous population is used as measure of this diversity: at the end of each generation, after the selection of chromosomes to carry in the evolution, average and standard deviation are computed on the fitnesses of the population, in order to pass these information to the successive population and manage the exploration.

Since the standard deviation alone cannot be representative for different situations, the reference measure ρ used by the heuristic rules is the ratio between standard deviation and average of the fitness:

$$\rho_G = \frac{\sigma_{fit,G}}{\mu_{fit,G}} \tag{5.2}$$

Where $G \in \{1,\ldots,MAXGEN\}$.

The heuristic rule for the strategy selection is:

$$strategy_{i,G} = \begin{cases} DE\,best & if\ \rho_{G-1} \le 5\cdot\rho_{\lim}\ and\ U_1(0,1) \le 0.7\ and\ U_2(0,1) < 0.5 \\ DERL & if\ \rho_{G-1} \le 5\cdot\rho_{\lim}\ and\ U_1(0,1) \le 0.7\ and\ U_2(0,1) \ge 0.5 \\ DE\,random & otherwise \end{cases} \tag{5.3}$$

Where $\rho_{\lim}$ is a user defined value used also in other heuristic rules and stopping criteria. This value is set as 0.005 and the recommended values are in the range [0.002; 0.01]. The two uniform random values selected for this rule are chosen for each i[th] chromosome, where $i \in \{1,\ldots,NP\}$.

Also the scaling factor F for the three strategies is selected following heuristic rules, in order to randomize the step length of the strategy adopted for each i[th] solution; this randomization is inspired by the *NSDE* approach. The randomization of F is performed transforming it into a Gaussian random variable with standard deviation σ and mean μ.

The standard deviation is set as 0.1, while the mean is defined by the following heuristic rule:

$$\mu_{F,i,G} = \begin{cases} 1.3 \cdot F & \text{if } \rho_{G-1} \le 4 \cdot \rho_{\lim} \text{ and } U_1(0,1] < 0.5 \\ 0.7 \cdot F & \text{if } \rho_{G-1} \le 4 \cdot \rho_{\lim} \text{ and } U_1(0,1] \ge 0.5 \\ F & \text{otherwise} \end{cases} \tag{5.4}$$

Then the final scaling factor is:

$$F_{i,G} = \begin{cases} \sim N\left(\mu_{F,i,G}, \sigma^2\right) & \text{if } U_2(0,1] < 0.5 \\ \sim N\left(-\mu_{F,i,G}, \sigma^2\right) & \text{if } U_2(0,1] \ge 0.5 \end{cases} \tag{5.5}$$

This rules previously described allow a sufficient diversification in the strategies: when the diversity in the population decreases, the greediness of *DE best* and *DERL* are used in order to speed up the exploration. Besides, the randomization of the scaling factor and its increasing or decreasing of 30% together, modify the step length, permitting at the same time exploration of unknown regions.

The stopping criteria

The stopping criteria adopted for this tool are three:

1. reaching of `MAXGEN = 80` generations
2. the best solution in the population is the same for `EQSOL = 10` generations consequently
3. the population diversity is under a specified value $\rho_{G-1} \le \rho_{\lim}$

The first and second stopping criteria affect the total optimization time: the first is difficult to reach, but the second is common for this classical situations.

Of course, this second one doesn't assure the attainment of the global optimum but, since the exploration abilities of the algorithm, 10 generations without any improvement seem a reasonable index of unexpected further evolution. The last criterion is directly referred to the population diversity: if it is too low, no more improvement could be done.

Since 20 seconds are approximately expected for each evaluation, the maximum time foregone is approximately 10 hours, considering a 20% of simulation failure for instability and unfeasibility of solutions:

$$t_{\max} = t_{individual} \cdot NP \cdot MAXGEN \cdot \eta = \\ = 12\left[\frac{s}{ind}\right] \cdot 20\left[\frac{ind}{gen}\right] \cdot 80[gen] \cdot 1.2 \cdot \frac{1}{3600}\left[\frac{h}{s}\right] \simeq 6.5\ h \tag{5.6}$$

This value is considered acceptable since the complex scenario under optimization.

The constraint's handling

Since the strict constraints, especially referred to the treating section, the elevated simulation time necessary for each solution and the high possibility to wasting time for unfeasible solutions, a particular shrewdness is adopted in order to resolve this highly constrained situation: a parallel population is introduced in order to store feasible solutions discarded by the selection process. The particularity introduced exploits the extreme greediness of the selection process for the next generation adopted by DE: the i^{th} evaluated child, called trial vactor, if feasible is compared one time only with the i^{th} vactor of the current population; if the trial one is infeasible it is automatically discarded. This selection feature excludes any feasible solution that unfortunately is compared with a fittest solution than it. Anyway, there is some possibility that this child is better than at least one of the current population; for this reason a parallel population is created, in order to store feasible solutions discarded by the main selection process. Of course this population has reduced dimension, comparable with the main population, and when a feasible solution is not accepted by the main population, it is compared with the worse of this parallel population: if the first is better than the latter, it takes its place. This procedure is inspired by the Preferential Crossover found in DEPC, Section 4.2.3.

This parallel population is than called to offer a feasible solution for the main selection process when the trial vector just evaluated results infeasible; in that case a second selection process is made between the i^{th} vector of the current population and a random selected solution from the parallel set. In that way the previously spent simulation time to evaluate a feasible solution is not completely wasted, giving to this initially discarded chromosome a second chance. This feature gives a good convergence speed to the evolution strategy since it keeps

relatively high the fitness level of the population, allowing high transmission of good properties to the next generations. This shrewdness shows its good properties after a latent period within also this population has to evolve: after this period the mean fitness values of the two populations are comparable and the second selection process keeps high the substitution in the main population.

The choice of this method for the constraint's satisfaction instead the penalty method or any repairing method is driven by the problem nature: since the oil produced grows together with the gas sent to the treating section, with high probability the optimum lies close to the infeasible area. The penalty method needs many generations to satisfy the constraint completely and a repair rule is not recommended, since the variables of the gathering system are not directly proportional to the oil production: a simple linear combination seems inapt.

5.2.6 Results

The optimization of the integrated asset is performed using three approaches:

1. a separated optimization of the two environments using the GAP internal optimization with the imposition of maximum gas rate. A successive optimization for HYSYS by DE is performed.
2. a separated optimization of the two environments using the GAP internal optimization with the imposition of maximum gas rate and FBHP limits. A successive optimization for HYSYS by DE is performed.
3. an integrated optimization of the entire asset (GAP + HYSYS) by DE, starting far from the optimized network solution.

The first optimization uses the optimization of the gathering system performed by the internal optimization tool of GAP. This optimization has as task the oil production increase and only one constraint: the maximum gas rate incoming to the process plant, since it is the well known problem of the asset. The maximum value allowable is 1450 kSm^3/day (Table 5.19). However, the GAP optimization manages only the choke opening, since the separators pressures are taken as end point pressures of the network, considered boundary conditions. This optimization removes two degrees of freedom to the entire system. The GAP results become then HYSYS inputs, from which starting the optimization of the process plant by the tool.

The second optimization uses another time the separated optimization, imposing the maximum gas rate of 1450 kSm^3/day and the minimum FBHPs, reported in Table (5.18). After this optimization by GAP, HYSYS is then optimized by the tool.

The third way exploits the integration concept, transforming the two environments into a unique system: in that case the superimposition of the

separators becomes unlocked and all the constraints, described in section 3 and resumed in Tab 5.18 and 5.19, could be satisfied contemporaneously. The gathering system variables are uniformly generated around the previous optimized values, while HYSYS is uniformly randomly generated inside ranges.

Tables 5.20 and 5.21 show the results for the three optimizations, the variables found in GAP for each well and in HYSYS and the constrained properties. It's important to notice the FWHP is not a variable but a simulation result, since it depends on the choke opening coupled with the downstream pressure (see Tab 4 and 5 for the constraints). This value indicates the production state of any well, the lower the FWHP, the higher the flow rate. Anyway, this pressure value is connected with the FBHP, a constrained property for the system.

Table 5.21, Variables and results for the gathering system for the three optimization: GAP optimization with gas rate constraint, GAP optimization with gas rate and FBHP constraints and optimization by the integrated tool with all the constraints.

GAP variables and results

	gas rate constraint				gas rate + FBHP				DE integrated tool			
	ΔP choke	FWHP	FBHP	Qoil	ΔP choke	FWHP	FBHP	Qoil	ΔP choke	FWHP	FBHP	Qoil
	bar	bar	bar	Sm3/day	bar	bar	bar	Sm3/day	bar	bar	bar	Sm3/day
W1	9.8	58.4	215	647	0.0	53.2	243.4	487	20.6	75.8	237.3	523
W2	14.0	44.0	199	1028	3.0	68.5	233.6	787	33.3	88.6	245.3	697
W3	no flow	no flow	324	0	6.8	92.4	274.7	1162	42.0	97.2	270.4	1253
W4	0.1	42.6	169	968	2.0	70.0	220.9	663	37.3	76.7	235.9	562
W5	1.0	55.0	237.5	386	0.0	44.2	252.7	296	0.4	48.5	256.7	272
W6	0.3	45.5	280.4	1434	1.1	56.9	279.6	1468	0.3	56.9	281.5	1392
W7	3.4	38.2	284	291	0.7	45.9	273.9	419	0.6	51.0	292.1	188
W8	0.1	35.0	231	347	0.0	50.8	257.9	210	0.1	36.2	235.1	327

Table 5.22, Variables, constraints and results in the HYSYS environment for the optimization 1,2,3. The separated optimization with the tool in HYSYS environment conceive the process constraints.

Separators		1	2	3
HP sep	[bar]	34	34	38.7
LP sep	[bar]	12.1	12	14.75
HYSYS parameters				
ΔP S1-S2	[bar]	1.7	1.1	1.0
P stab	[bar]	9.5	9.5	9.5
T bottom	[°C]	186	188	188
T heat-exch	[°C]	115	100	80
HYSYS constraints				
Inlet oil	[bbl/day]	35336	35184.91	36910
Inlet gas	[kSm3/day]	1044	1206	1408
Inlet water	[Sm3/day]	384	504	257
Gas treated	[km3/day]	38.4	42.288	41.9
CO2/H2S	[-]	7.6	15	18.1
Wobbe	[MJ/m3]	52	50.5	51.7
TVP	[kPa @100F]	85	84.3	85.6
HYSYS results				
Qoil out	[bbl/day]	35019	34249.06	35940
Qgas out	[kSm3/day]	1007	1008.24	1145

The optimization times are reported in Table 5.22.

Table 5.23, Optimization times for the three optimizations.

	GAP time	**HYSYS time**
GAP gas rate	45 min	20 min
GAP gas rate + FBHP	1 h	20 min
Integrated tool	~4.5 h	

Figures 5.23 and 5.24 show the resulting FWHP of the solutions of the three optimizations and the oil production achieved. The FWHP is one of the most important properties of the system from a production point of view, since is easily measurable.

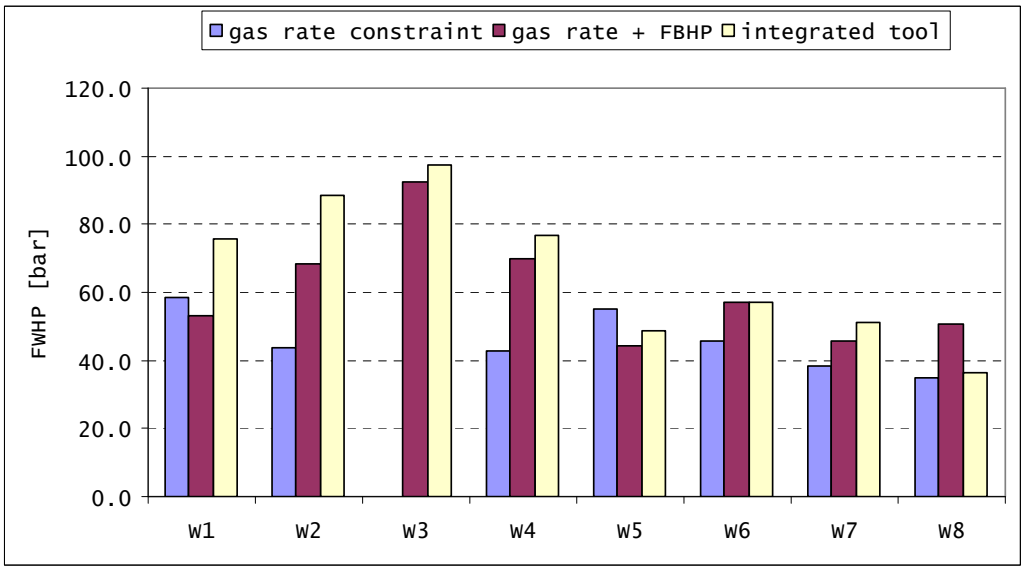


Figure 5.38, FWHPs resultant from the three optimizations. The values are similar except for W3, completely closed by the first optimization. This well has the highest GOR in the field's line.

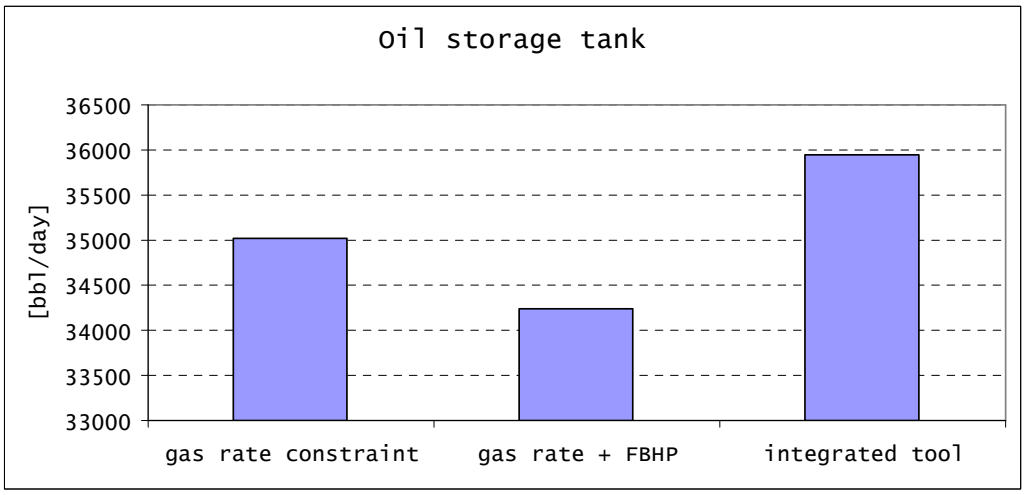


Figure 39, Oil production achieved by the three optimizations. The last one is approximately 3% higher than the first.

As expected the first optimization, performed separately for the two environments, is not able to satisfy all the constraints: three FBHP are under the minimum values allowable, in particular for the wells W1, W2 and W4. The W3 is completely close, since its high GOR and sourness. This situation is completely unsatisfactory, since W3 has high GOR but unfortunately also a good well performance. The optimization of the HYSYS environment does not give considerable improvements on the oil production but it works mainly on the satisfaction of its constraints. At the end the tool applied only to the process

environment, at least for this particular case, is not significant for the global optimization.

The second optimization, that should be more reliable in terms of constraints' satisfaction, finds a worse solution: the oil production in this case is approximately 2% less than the previous optimization. As clear from Figure 5.23, the FWHP setting is quite different respect to the previous one, but the final oil production is diminished. W2 and W4, in the first optimization do not satisfy the FBHP constraint. With the second optimization these two wells are induced to produce less in order to obtain the satisfaction of their bottom-hole constraint. For this reason their missed production reduces the final oil production, even if W3, in this configuration, starts to produce. Farther, one constraint is not satisfied in the process plant: the gas sent to the treating section goes slightly beyond the limit of 42 m^3/day imposed.

The last optimization, even if it takes 4.5 hours, satisfies the constraints of the entire system and at the same time it enhances the final oil production. This value is approximately the 2.7% higher than the result obtained with the first optimization. This result is achieved manipulating the separators' pressure, variables considered fixed for the separated optimization. Increasing the first separator pressure, the amount of gases released from the separators' stages has a lower flow rate. In fact, the most important constraint in the plant is the gas flow rate to the treating section; its limitation is referred to the actual gas flow: the higher the pressure, the lower is the volume flow keeping the mass rate constant. For this reason, the first separator pressure is increased, approaching the upper boundary for this variable. Also the limit of this constraint is approached: in fact, this property of the system is the bottle-neck of the entire production chain for this line.

The ability of this tool to explore regions close to the boundaries of the feasible region is higher respect to a separated optimization. Combining the two environment into a single one assures the attainment, at least with the heuristic implementation described in Section 5.2.5, of the global optimum of the system. Other runs are performed and the solutions obtained are the same as reported in this section, proving the reliability and robustness of this tool.

5.3 A real case study. A nuclear safety system: multi-objective optimization of inspection intervals.

In this case study we consider the problem of the optimization of the inspection intervals of a nuclear safety system. For its solution, we investigate the use of DE and compare it respect to other evolutionary algorithms, already presented in Section 5.1. In the comparison, we look in particular at the computation time and at the characteristics of the Pareto frontier. The problem is first treated as a SO optimization and then as a MO optimization.

It refers to the choice of the time intervals for the periodic testing of the components of the High Pressure Injection System (HPIS) of a Pressurized Water Reactor (PWR).

Reliability/availability design and inspection/maintenance strategies are the target to optimize for these safety systems. Because of their difficult complete understanding on a global view, the complexity of the interactions between different objectives and the difficulty to evaluate their impact under a unique measure, the problem represent one of the most interesting cases for the multi-objective optimization in nuclear industry.
Typically, this kind of systems is subject to physical and normative constraints which come into play imposing restrictions that the candidate solutions have to satisfy. For simplicity, in our case studies we do not impose any a priori constraint to be satisfied by the candidate solutions.

5.3.1 The problem

We tackle the issue of finding the inspection intervals for the components of the HPIS (High Pressure Injection System), a safety system for nuclear power plant for a Pressurized Water Reactor, which intervenes in case of a small LOCA (Loss of Coolant Accident). The optimization is sought with respect to different conflicting objective functions: (i) the mean availability, (ii) the cost of the inspections (and the eventual cost of repair in case of accident) and (iii) the exposure time of the maintenance operators.

For this study the following assumptions are made:

1. At least one of the flow paths must be open at all times.
2. If the component is found failed during surveillance and testing, it is returned to an as-good-as-new condition through corrective maintenance or replacement.
3. If the component is found to be operable during surveillance and testing it is returned to an as-good-as new condition through restorative maintenance.
4. The process of inspection and testing requires a finite time; while the corrective maintenance (or replacement) requires an additional finite time, the restorative maintenance is supposed to be instantaneous.

The simplified scheme of the system is shown in Figure 4, as already presented and explained in [17].

The three objective functions are computed on the basis of a classical fault tree and event tree analysis.

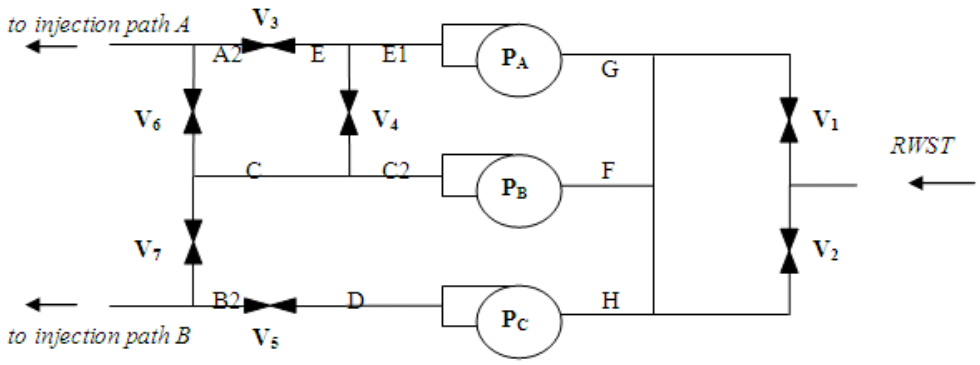


Figure 5.40, HPIS simplified scheme [17]

The system components are divided in three groups; all the items belonging to a same group undergo testing with the same periodicity. The three inspection intervals are identified by T_i, i = 1, 2, 3. The maintenance item groups are:

T_1 → {V_1, V_2}
T_2 → {V_3, V_5, P_A, P_B, P_C}
T_3 → {V_4, V_6, V_7}

The groups contain respectively the inlet valves, the pumps together with the outlet valves and the crossover valves.

$\underline{T}$ = [T_1 T_2 T_3] is the decision variable array composed by the three inspection times referred to each maintenance item group. The reference time is one year, and the time inspection variable is expressed in hours, so the domain is $1 \leq T_i \leq$ 8760 hours, i = 1, 2, 3.

The test interval specified by the technical specifications (TS) both for pumps and valves is 2184 h. So, for the previous case, the variable array recommended by TS is $\underline{T}$ = [2184 2184 2184] h.

Since we prefer speak in terms of minimization, the maximization of the availability is replaced by minimization of the mean unavailability.

The three objective functions are defined by models present in literature.

The mean unavailability is computed after the determination of the fault tree for the top event “no flow out of both injection paths A and B”. The resulting minimal cut sets then are reported in Table 5.23.

The mean unavailability can be expressed as follows:

$$\bar{U} \approx \sum_{j=1}^{N} \prod_{i=1}^{n_j} \bar{u}_i^j \tag{5.7}$$

where N is the number of minimal cut sets, n_j is the number of basic events relevant to the j^{th} minimal cut set and $\bar{u}_j^i$ represents the mean unavailability associated with the i^{th} component of the j^{th} minimal cut set.

Table 5.24, Minimal cut sets for the safety system reported in Figure 5.25.

MCS #	Components	order
1	V_1, V_2	2
2	V_5, P_A, P_B	3
3	P_A, P_B, P_C	3
4	V_3, V_4, V_5, P_B	4
5	V_3, V_4, P_B, P_C	4
6	V_3, V_5, V_6, V_7	4
7	V_3, V_6, V_7, P_C	4
8	V_4, V_5, V_6, V_7, P_A	5
9	V_4, V_6, V_7, P_A, P_C	5

For mean unavailability of a generic component *I*, several models have been proposed in literature. In this study the model used is:

$$\bar{u}_j^i = \rho_i + \frac{1}{2}\lambda_i T_i + \left(\rho_i + \lambda_i T_i\right)\frac{d_i}{T_i} + \frac{t_i}{T_i} + \gamma_0 \tag{5.8}$$

where ρ_i is the probability of failure on demand, λ_i the failure rate of the i^{th} component, T_i the test interval, t_i the mean downtime due to testing, d_i the mean downtime due to maintenance and γ_0 the probability of human herror. Eq. (5.8) is valid when $\rho<0.1$ and $\lambda T<0.1$, which are reasonable assumptions.

The cost function is composed by two contributions:

1. $C_{S\&M}$: cost for surveillance and maintenance
2. $C_{accident}$: cost associated with consequences related to accidents

Then the cost is:

$$C = C_{S\&M} + C_{accident} \tag{5.9}$$

The cost S&M is so defined:

$$C_{S\&M} = \sum_{i=1}^{N_C} \left[C_{ht,i} \left(\frac{T_M}{T_i} \right) t_i + C_{hc,i} \left(\frac{T_M}{T_i} \right) d_i \left(\rho_i + \lambda_i T_i \right) \right] \tag{5.10}$$

where $C_{hc,I}$ is the yearly inspection cost, $C_{ht,I}$ is the corrective maintenance cost and T_M is the mission time, 8760 hours in our case.

The accident cost is intended as a measure of the cost associated to damages of accident which are not mitigated by HPIS intervention. Using a small LOCA event tree found in literature [61] and reported in Figure 5.26, the following formulation describes the accident cost:

$$\begin{aligned} C_{accident} &= C_1 + C_3 \\ C_1 &= P(EI)(1-U_{RT})\bar{u}\left[U_{LPIS} + (1-U_{LPIS})U_{SDC}U_{MSHR}\right]C_{PDS1} \\ C_3 &= P(EI)(1-U_{RT})\bar{u}\left[U_{SDC}(1-U_{MSHR}) + (1-U_{LPIS})\right]C_{PDS3} \end{aligned} \tag{5.11}$$

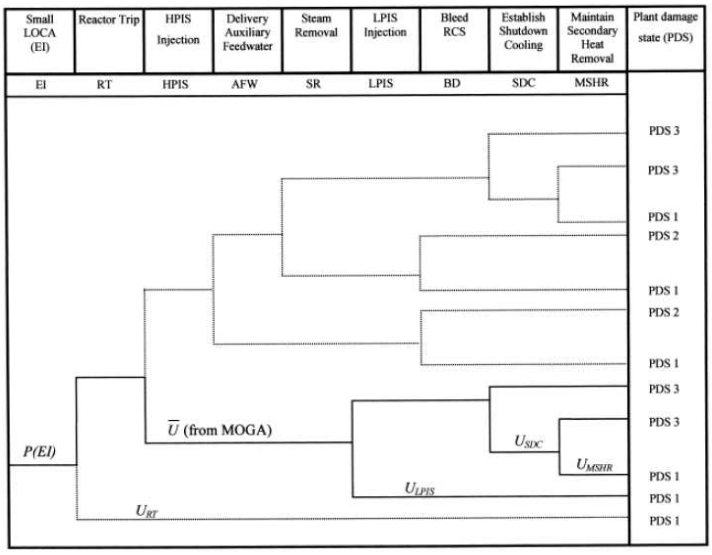


Figure 5.41, Event tree for the initiating event small LOCA [61]

These costs depend on the initiating event frequency and on the unavailability values of the safety systems which ought to intervene along the various sequences: these values are taken from the literature [61].

During testing operations, the technicians may be subjected to radiation exposure: based on the well-known ALARA (As Low As Reasonably Achievable) and limit-dose principles, the dose received by workers should be minimized. Assuming a constant exposure rate, the minimization of the dose is

equivalent to that of the exposure time, so that the third objective function could be formulated as:

$$T_{\exp} = \sum_{i=1}^{N_c} \left[\left(\frac{T_M}{T_i} \right) t_i + \left(\frac{T_M}{T_i} \right) d_i \left(\rho_i + \lambda_i T_i \right) \right] \tag{5.12}$$

The first objective function is in conflict with the other two, since frequent inspection times tend to small mean unavailabilities but increase the costs and the exposure times. For this reason, the multi-objective optimization is treated in terms of the concept of dominance of solutions, seeking for Pareto frontier.

5.3.2 The optimization schemes

Three optimization schemes are adopted in this paper:

1. single-objective constrained optimization
2. weighted sum scheme
3. MO with dominance concept.

They are used in this paper to investigate the strengths and the weaknesses of *MODE* (*Multi-Objective Differential Evolution*), a tool developed in MATLAB by LASAR (LAboratory of Signal and Risk Analysis http://lasar.cesnef.polimi.it/) of the Energy Depratment of Politecnico di Milano, provided with the SO and MO options. Several variants are implemented in the tool, in order to increase its flexibility and ability to tackle different problems. For MO option *MODE-III* is implemented.

Two algorithms are used for comparison:

1. *Genetic Algorithm toolbox*
2. *Multi-Objective Genetic Algorithm MOGA,*

Genetic Algorithm toolbox (*GA-toolbox*) has several sophistications and internal variants. A complete descriptive help is available on the program and online. When no particular settings are imposed to this tool, many of the sophistications implemented are used with default setting. Anyway, the correct usage for specific problem needs deep consciousness of the tool.

Also *MOGA* has several variants adoptable, and the number of information necessary to its running is high. Further, a wrong strategy selection could provoke failure of the optimization. Also for this tool the setting is not easy.

Single-objective constrained optimization

The optimization of the inspection intervals T_1, T_2 and T_3, is first tackled as a SO constrained optimization, where the mean unavailability is optimized with cost as constraint, and vice versa. Since the third objective function, the

exposure time, has the same proportional dependence from the inspection times as the cost, it is ignored in this preliminary single-objective optimization.

The constraint values are taken from the technical specifications that recommend $\underline{T}$ = [2184 2184 2184] hours. This gives mean unavailability and cost constraints:

- $U_{constr} = 3.5427 \cdot 10^{-4}$
- $C_{constr} = 1440.2$ \$

For this optimization, *MODE* and *MOGA* are compared in their SO version.

The SO mutation variants adopted for this optimization with *MODE* are:

- *DE random*
- *DE best*
- *DERL* (DE with Random Localization)
- *NSDE* (Neighbourhood Search DE)
- *TDE* (Trigonometric DE)

These variants are considered the most reliable and fast modifications of the original DE. Their strengths and weaknesses have been shown in Section 5.1. Since the optimization is constrained, *DE* is coupled with a repair technique for infeasible solutions. In fact, if a solution is infeasible, it must be discarded. This moving is obtained with a bisection method applied between feasible and infeasible solutions.

This practice assures less total function evaluations.

Weighted sum scheme

A second optimization is performed by adopting the weighted sum scheme of the three objectives to reduce the multi-objective optimization to a single-objective one: this leads to the identification of only one solution, highly depended on the weights.

As for the first test, the five DE variants above are tested.

The overall function that integrates the three targets into one is so defined:

$$f(\underline{T}) = w_U \cdot U_n(\underline{T}) + w_C \cdot C_n(\underline{T}) + w_{T_{\exp}} \cdot T_{\exp,n}(\underline{T}) \tag{5.13}$$

where the *n* subscript is referred to a normalized value.

Each objective function is normalized with the following rule:

$$f_n = \frac{f - f_{\min}}{f_{\max} - f_{\min}} \tag{5.14}$$

where $f_n = U_n, C_n, T_{\exp,n}$ and $0 \le f_n \le 1$.

The sum of the three weights w_f , f = U, C, T_{exp}, must be 1, then also $0 \le f\left(\underline{T}\right) \le 1$.

The maximum and minimum values taken are:

$$0 \le U \le 7.5 \cdot 10^{-4}$$

$$0 \le C \le 2000 \ [\$]$$

$$0 \le T_{\exp} \le 200 \ [h]$$

The normalization is necessary because the objective functions have different orders of magnitude.

In this optimization 10 weights' settings are tried and the results are compared to the Pareto frontier obtained by *MOGA*.

Multi-objective optimization with dominance concept

The third optimization is performed with a multi-objective Pareto approach. The objective functions are maintained separate and the Pareto dominance concept is used to define the Pareto front of equally good solutions.

The multi-objective optimization of the safety system inspection is tested with *MODE*. The resulting Pareto frontier and the times to reach it are compared with the results of the other two algorithms:

1. *GA-toolbox* for MO optimization
2. *MOGA*, already used as reference in the previous approach

MODE has three variants allowable in MO options:

a. *DE random*
b. *NSDE*
c. *SACPDE* (Self-Adaptive Control Parameters for DE)

The choice of these variants for the implementation of *MODE* is principally driven by the absence of fitness feedback in their reproduction phase, for the sake of the speed of the algorithm. In MO optimization, the superiority of a solution could be defined only after a dominate comparison ranking of all solutions in the population, and this slows down the algorithm.

Thanks to the powerful recombination process of *MODE-III* and its greedy selection between parents and trial solutions for the successive generation, dominate comparison and ranking of all solutions are not necessary, saving computation time.

5.3.3 Results

5.1 Single-constrained optimization

The optimization results obtained by *MOGA* are showed in Table 5.1 [17].

Table 5.25, *MOGA* results on single-objective optimization of inspection intervals [17]

	U optimization $C \leq C_{constr}$	C optimization $U \leq U_{constr}$	
T_1	549	1672	[h]
T_2	2852	8742	[h]
T_3	5492	8246	[h]
U	**$2.3208 \cdot 10^{-4}$**	$3.5187 \cdot 10^{-4}$	[-]
C	1436.2	**529.3**	[$]

The stopping criteria adopted in the DE search are

- $\Delta=|f_{min}-f_{max}|$ of the current population is less than eps, practically the whole population is converged at the same point if eps is sufficiently small compared to the fitness' order of magnitude
- reached MAXGEN generations

The optimization performed with *DE* has the following general parameters:

```
Population Size NP         30
Maximum Generation MAXGEN  500
eps U/C               1e-8/1e-4
```

Because of the different orders of magnitude of the mean unavailability and the cost, to make the first stopping criterion efficient the `eps` values must be different.

The settings for the *DE* variants are:

- *DE random* `CR=0.7, F=0.5`
- *DE best* `CR=0.7, F=0.5`
- *DERL* `CR=0.7, F=0.5`
- *NSDE* `CR=0.7, NS=0.5`
- *TDE* `CR=0.7, F=0.5, MT=0.05`

The choice of the previous parameters' settings is mainly driven by the author experience.

The crossover rate *CR* affects the amount of perturbation introduced in the trial vector from the noisy one: if this value is high, or close to 1, the trial vector is practically the noisy vector, if it is small, the trial vector inherits high fraction of its variables from the target vector instead from the noisy. So, if *CR*

is small, the trial vector has small differences respect to the target and the convergence speed is small. Otherwise, if *CR* is high the convergence speed is high but the possibility to be trapped in a local optimum increases. Since the two objective functions, mean unavailability and cost, are expected with low local optima and the dimensionality is low (only three variables), the value 0.7 for *CR* is considered suitable. Nevertheless, if the objective function to optimize is complex and with several local optima or the dimensionality increases, small *CR* values are recommended (*CR*<0.3) (see Section 5.1.1).

The scaling factor setting is significantly less critic than the crossover rate setting. The value $F = 0.5$ for *DE random*, *DE best* and *DERL* is taken, because in the middle of the recommended range (0, 1].

NS controls the *NSDE* approach: if NS is high the search is more concentrated in the neighbourhood, whereas if NS is small the Cauchy operator, characterized by high values, enhances the search space exploration. A correct mediation between these two searching methods is achieved by NS=0.5.

TDE approach, which uses the *DE random* scheme as basis reproduction phase, has the same *F* and *CR* of the first variant; the mutation probability with the trigonometric mutation M_t is set as 0.05: this value permits a demonstration of *TDE* ability in this problem. If M_t is too low the behaviour of this scheme approaches the *DE random* behaviour.

Tables 2 and 3 report the results for the first two optimization schemes, repeated 50 times in order to obtain significant statistical values with respect to the randomness. To demonstrate the accuracy when required, the standard deviation is also reported. Figures 6 and 7 report the convergence speed in terms of mean function evaluations (fe) and mean cpu time (cpu) used to complete each single run for the five tested variants. The number of function evaluations is the product between the population size and the number of generations performed, plus the value of additional evaluations for the repair rule.

Table 5.26, Mean unavailability results by DE optimization. Cost is the constraint

	DE rand		DE best		DERL		NSDE		TDE	
	mean	Std	mean	std	Mean	Std	mean	std	Mean	std
fe	1597	210	1215	343	1288	197	1973	327	1617	314
cpu	0.1847		0.1406		0.1734		0.2284		0.2078	
U	**2.3203E-04**		**2.3205E-04**		**2.3204E-04**		**2.3203E-04**		**2.3203E-04**	
T1	543.8	1.8	544.3	2.7	544	1.9	544.2	1.6	543.8	2.6
T2	2862.3	11.1	2863.9	27.7	2864	17.9	2859	11.7	2863.3	15.2
T3	5325.6	189	5311.9	570.9	5285.8	283	5366.9	213.5	5309.1	257.3
C	1440.2		1440.2		1440.2		1440.2		1440.2	

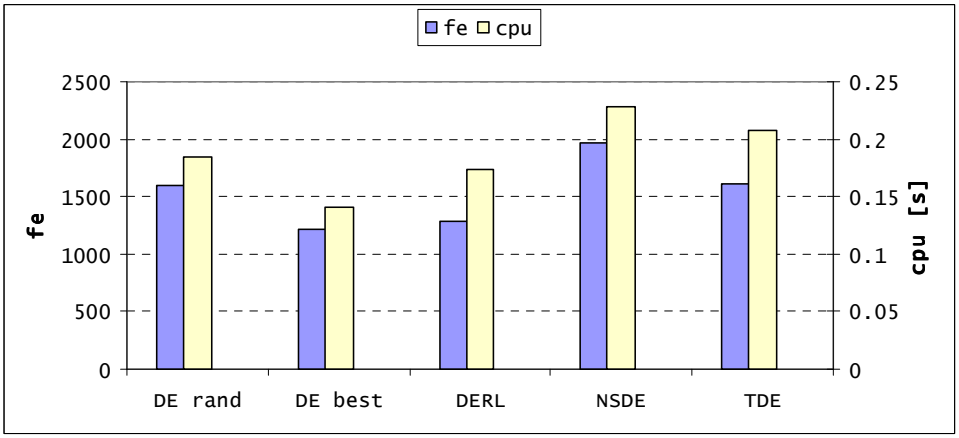


Figure 5.42, Function evaluations and cpu-time used for *U* optimization by DE variants.

Table 5.27, Cost results by DE optimization. Mean unavailability is the constraint.

	DE rand		DE best		DERL		NSDE		TDE	
	mean	std	mean	std	Mean	std	mean	std	Mean	std
fe	2489.6	290.6	1230	137	1840	217	2899	344	2557	332
cpu	0.2884		0.1453		0.2513		0.34		0.3316	
U	3.5427E-04		3.5427E-04		3.5427E-04		3.5427E-04		3.5427E-04	
T1	1691.9		1691.9		1691.9		1691.9		1691.9	
T2	8760		8760		8760		8760		8760	
T3	8760		8760		8760		8760		8760	
C	**524.1**		**524.1**		**524.1**		**524.1**		**524.1**	

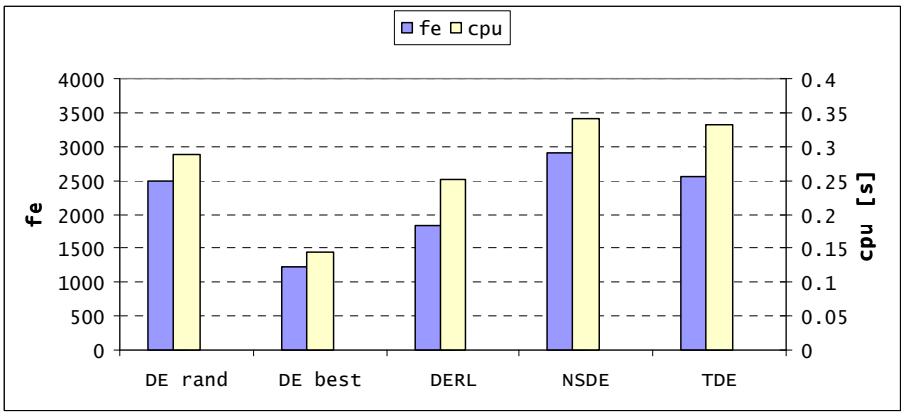


Figure 5.43, Function evaluations and cpu-time used for *C* optimization by DE variants.

In the SO constrained optimization, *MODE* outperforms *MOGA* in both cases of cost and variables constraints.

MOGA finds $U = 2.3208 \cdot 10^{-4}$ and $C = 529.3$ \$ while *DE random*, *NSDE* and *TDE* find $U = 2.3203 \cdot 10^{-4}$ and all the variants find $C = 524.1$ \$, just at the unavailability constraint limit of $3.5427 \cdot 10^{-4}$.

Achieving the same minimum value at the constraint limit shows the high reliability of the 5 DE variants.

NSDE is accurate but the number of function evaluations is high, due to its Cauchy scaling factor: using high *F*, the probability to find an infeasible solution is high and repair is required.

DE best and *DERL* are the fastest but less accurate variants: their behaviours are reflected by the standard deviations for the inspection intervals results: T_3 is the less significant variable (its standard deviations are large), while T_1 and T_2 have more impact on the unavailability; *DE best* and *DERL* show the two largest standard deviations for T_2, sign of their inferior accuracy. For T_1, these values are generally small. *DE random* and *TDE* are the most accurate variants, but *TDE* has higher computation time than *DE random*, even if the number of function evaluations is the same: this because *TDE* needs a high information processing, not justified in this case.

As for the unavailability minimization, all the variants approach the same solution in terms of variables' values and consequently of objective function and constrained limit (see Table 3). Also for the cost minimization, the DE is shown to be reliable in all variants.

The difference is again in the convergence speed (Figure 7): *DE best* is the fastest with the smallest standard deviation. This means that the cost function has a simple shape that exalts the search abilities of *DE best*. *DERL* has the second fastest convergence speed, while *DE random* and *TDE* have the same behaviour as in unavailability optimization: same function evaluations but highest cputime for the second. *NSDE*, as in the previous case, has the highest number of function evaluations and consequently of cputime because the optimization is constrained and unfeasibility arise.

Weighted sum approach

This approach transforms a MO optimization into a SO one by the weighted combination of the multiple objective functions. The setting of these weights moves the optimization toward a specific objective function. Changing the weights, each run returns a point that lies on the Pareto frontier. To obtain a dense Pareto front, number of weights' settings must be used. In our test, ten weight settings are tried, returning only ten solutions.

The three weights must be chosen coherently: since the second objective function has the same proportionality with respect to time as the third one, the second and third weights are set equal; the first weight is set at 10 different values in a range between `0.05` and `0.95`. Figure 8 shows, in two dimensions (*U* and *C*), the *MOGA* Pareto frontier and the ten points obtained

from the weighted-sum method optimized by DE. The five variants approach to the same solutions in all the weights' settings since the high reliability of DE on finding the true global optimum; only the computation times are different.

The stopping criteria adopted are the same as for SO optimization.

The total number of function evaluations (fe) and cputime spent for the ten runs are presented in Table 4.

The algorithm parameters are:

NP	100	
MAXGEN	500	
eps	1e-6	
CR	0.5	
F	0.5	
NS	0.5	(only for *NSDE*)
MT	0.05	(only for *TDE*)

The Pareto front used as comparison is obtained by a *MOGA* run with parameters NP=100 and MAXGEN=500.

The adoption of a lower value of CR, with respect to the previous SO optimization, is justified by the diversity of the objective functions: the contemporary search in the minimization of three objective functions with high CR could unbalance the results, inspite of the difference in the weights.

The ten points obtained by the weighted-sum scheme lay on the Pareto frontier, but the definition of minimum and maximum values for normalization of the objective functions limits the searching area. In fact, the result obtained with w_U = 0.05, w_C = 0.475, w_T = 0.475 (biasing the target on the cost and exposure time minimization) is close to the maximum value of U equal to $7.5 \cdot 10^{-4}$ (highest U, lower C). Nonetheless, this behaviour is not found for an opposite weights' setting biasing toward an optimization of unavailability (w_U = 0.95, w_C = 0.025, w_T = 0.025): the cost does not reach its maximum value of 2000 $ but it finds a solution near 1300 $.

The weighted-sum scheme makes it difficult to obtain a good convergence of the Pareto frontier since it is too sensitive to the weights' setting.

Table 5.28, Function evaluations and cputimes for the five variants tested in weighted sum-scheme

	fe	Cpu [s]
DE random	37109	2.92
DE best	20757	1.78
DERL	27923	2.95
NSDE	43252	3.36
TDE	36369	3.58

DE best is the fastest algorithm, both in fe and cpu time; *DERL* follows in fe but the cputime is comparable with *DE random*; this reflects the absence of fitness feedback information processing, proper of *DERL* and *TDE*.

TDE is the variant with the highest cputime because of its high information processing. *NSDE* also in this situation fails: its high capability to explore wide searching area brings it out of boundaries. *DE best* outperforms all the other variants.

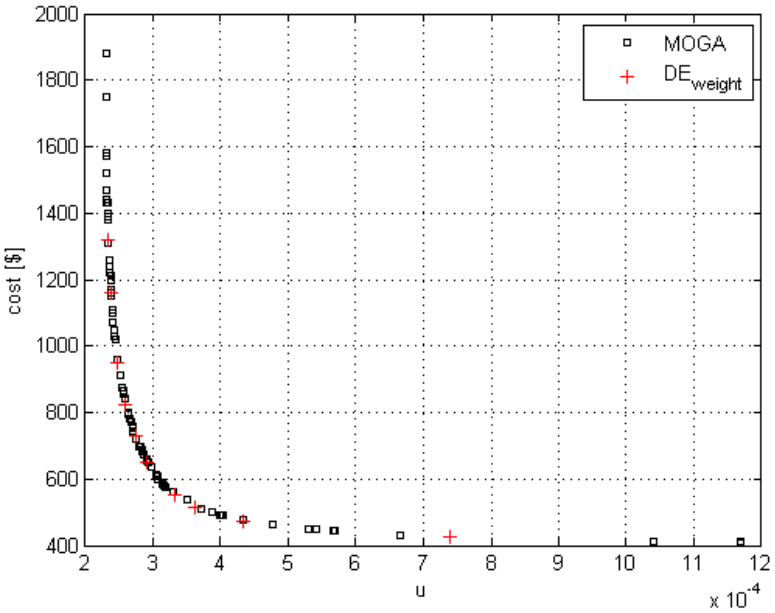


Figure 5.44, Ten solutions obtained with ten different settings on weighted-sum scheme applied to *DE*, compared with *MOGA* Pareto frontier

Multi-objective approach

The population size for the three algorithms (*MOGA*, *GA-toolbox* and *MODE*) is fixed as NP=200 and the stopping criterion is set as the attainment of MAXGEN=500 generations. Then the number of total function evaluations is the same for the three algorithms (10000).

GA-toolbox and *MOGA* distinguish in two parameters the number of non-dominated solutions set in the final archive and the population size: for them both are set equal to 200.

Of course, *MODE* has some disadvantages in terms of the density of the dominant set, since it carries on only *NP* individuals in a unique archive that is skimmed only at the end of the run: only if the whole population reaches the Pareto frontier the number of non-dominated points will be equal to 200, otherwise the population is skimmed.

Figures 9 and 10 show the Pareto front achieved by the three algorithms: for *MODE* the version *DE random* is plotted on the left, while on the right the non-dominated solutions found by the three *MODE* variants are shown.

The three algorithms approach to the same Pareto frontier, but with different densities and boundaries; *MOGA* and *MODE* seem to be more reliable in the Pareto frontier search for low costs and exposure times than *GA-toolbox*. In fact, the latter concentrates its search in the low mean unavailability region, but it does not explore the space beyond 2000 $ of cost (see Figure 10).

Table 5 reports the time for completing the search and the number of solutions in the Pareto set found by the algorithms. The three *MODE* variants take approximately the same time, since they compute the same amount of function evaluations (`10000`) in the same programming environment.

Table 5.29, Cputime and number of Pareto solutions present in the final archive for the *MOGA*, *GA-toolbox* and the three *MODE* variants

Variant	cpu	NP
MOGA	~10 min	200
GA-toolbox	~2 min	200
MODE, DE random	5.672 s	148
MODE, NSDE	6.328 s	145
MODE, SACPDE	6.109 s	153

The number of the non-dominated solutions found by *MODE* is not the same as for *MOGA* and *GA-toolbox*: this is because only a fraction of the population reaches the Pareto frontier. However, the computation times are significantly smaller than those of the other two algorithms.

The cputime comparison between *GA-toolbox* and *MODE* is particularly interesting as both are implemented in Matlab: the latter algorithm is about 20 times faster than the first.

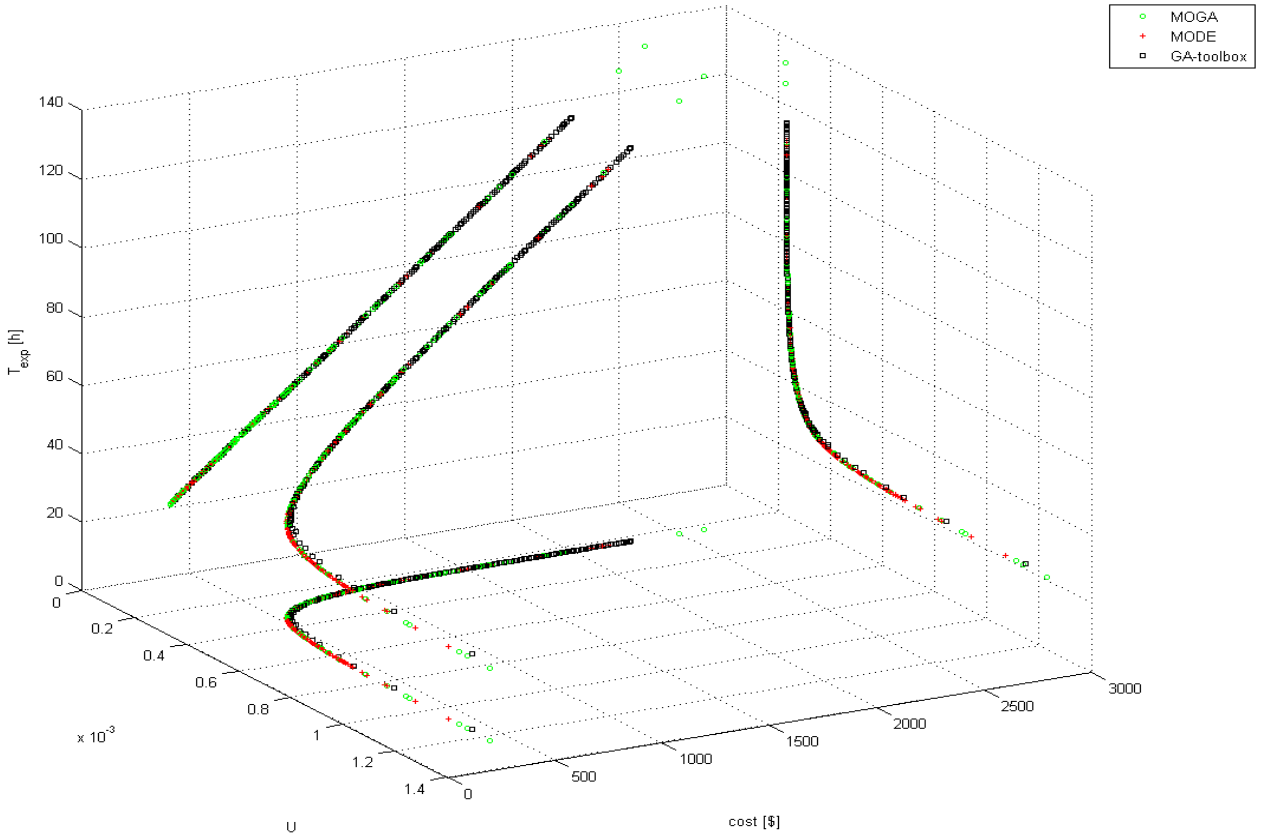


Figure 5.45, The Pareto fronts obtained by *MOGA*, *MODE-random* and *GA-toolbox* in the inspection intervals optimization

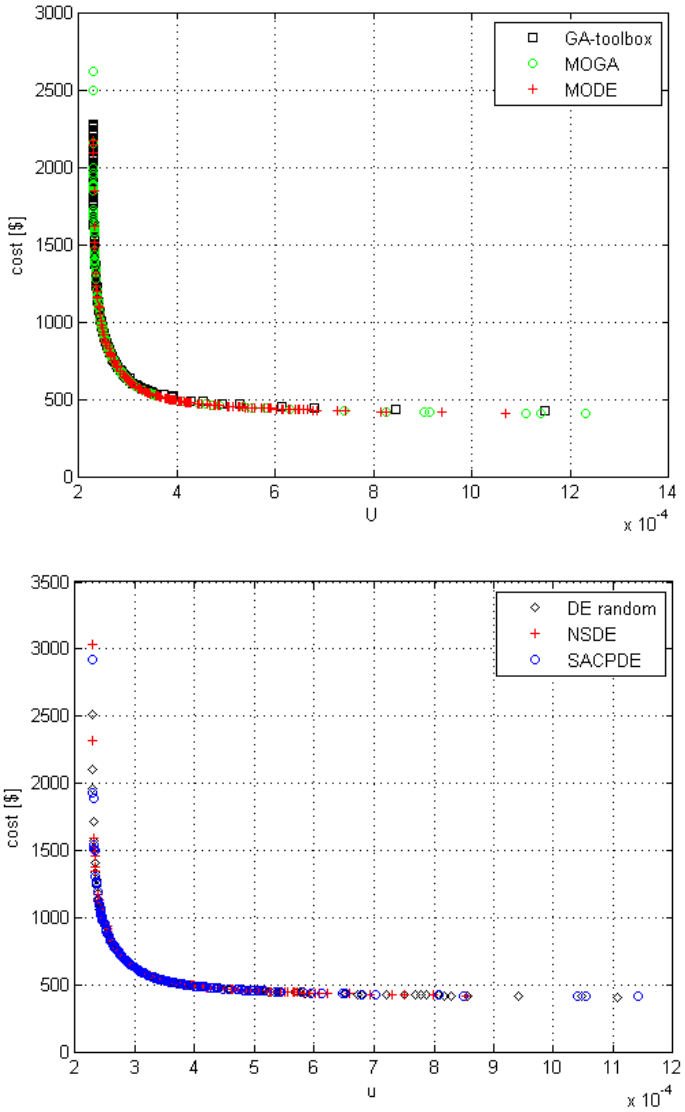


Figure 5.46, The Pareto frontiers of Figure 9 in two dimension: *U* and *C* for *GA-toolbox*, *MOGA* and *MODE-random* on the top and the Pareto frontiers for the three *MODE* variants on the bottom

These differences in computation time depend on the complexity of the algorithm: in this case the simplicity of DE is rewarded: the Pareto front is satisfactory and the percentage of non-dominated solutions is remarkable.

MOGA seems to be reliable but really slow.

Table 5.30, Direct comparison between the three algorithms and the three *MODE* variants for inspection intervals optimization

	MODE-rand	MODE-NS	MODE-SACPDE	GA-toolbox	MOGA
MODE-rand	-	21.85	27.73	25.45	10.92
	-	27.69	13.43	10.91	15.13
MODE-NS	22.58	-	28.23	27.73	5.65
	18.55	-	26.61	12.61	15.32
MODE-SACPDE	19.3	26.32	-	55.26	5.26
	23.68	25.44	-	11.4	15.79
GA-toolbox	24.5	11.5	20.5	-	27.5
	9	9.5	18	-	8
MOGA	21	26	30	25	-
	12	10	7	29	-

Table 6 shows the percentage of superiority and inferiority of one variant against another one referred to its solutions: the variants of the rows are compared with the variants of the columns and the first number in each cell represents the percentage of dominant points of the row that dominate over the column's algorithm solutions, while the second one represent the fraction of the dominated points (e.g.: *MODE-random* has 21,85% of its Pareto set that dominates *MODE-NSDE* front, while the 27.69% of *MODE-random* points are dominated by some *MODE-NSDE* points – first row, second column).

Except for *MODE-SACPDE* that has 55% of its frontier that dominates that of GA, the other percentages of dominant and dominated solutions are relatively small, around 20-30%. Moreover, the values of dominant and dominated fractions of an algorithm are similar, making it difficult to declare superiority of one over the other.

Also the three *MODE* variants do not show clear superiority: no improvements are carried by self-adaption of parameters (*SACPDE*) or by the neighbourhood search (*NSDE*).

On the other hand, relevant differences remain in terms of computation time, driven principally by the differences in parameter setting: *DE random* keeps constant during the optimization the two parameters *F* and *CR*, while *SACPDE* and *NSDE* need new parameters' generations. *SACPDE* applies the evolution of parameters only sometimes, when the heuristic rule is satisfied, while *NSDE* generates new scaling factor for each individual for any generation; this is the reason of the different cpu times.

5.3.4 Conclusions

The results obtained in this paper represent an improvement in the optimization of complex nuclear system as the safety system studied. The novel evolution algorithm tested, DE, works well and fast with respect to the MO optimization version of GA, even with different strategies.

In the single-objective optimization, *MODE* outperforms *MOGA* in terms of accuracy of the solutions, reaching lower values for mean unavailability and cost respectively. In that case *DE best* variant is considered the fastest algorithm but it should be used carefully because it could be less accurate. *DE random* remains the most robust and reliable variant if no information about the problem are available.

The weighted sum scheme applied to *MODE* permits the achieving of the same Pareto frontier obtained by *MOGA* in MO option, but this approach is extremely dependant on the weights used for the integration of the three targets into one. For this reason this approach is considered not satisfactory, since a previous knowledge of the problem nature is necessary for the weights' setting. No particular conclusion could be done respect the accuracy; only the time is comparable and also this time *DE best* is the fastest since its greediness.

For the MO non-dominance approach, *MODE* outperforms *MOGA* and *GA-toolbox* only in terms of convergence speed, which is significantly high: *MODE* is 100 times faster than *MOGA* and 20 times than *GA-toolbox*.

The convergence speed of this tool is its main advantage, thanks to its simplicity. The other evolutionary algorithms need sophistications and difficult parameter settings, while *MODE* has very few parameters. Of course the number of solutions on the Pareto frontier is not the same as the population size, while the other algorithms may achieve the desired number of points of the Pareto front by storing the dominant solution in an archive uploaded generation by generation.

Anyway, the number of non-dominated points carried to the Pareto front (75%) is satisfactory.

References

[1] D.E. Goldberg, “Genetic algorithms in search, optimization, and machine learning”, Addison-Wesley Publ. Co., 1989.

[2] Nelder, J. A., and R. Mead. 1965. “A simplex method for function minimization”. Comput. J. 7:308–313.

[3] J.E. Dennis and V.J. Torczon. SIAM J “Optimization” 1(4): 448-474,1991

[4] Kirkpatrick, S., C. D. Gelatt Jr., and M. P. Vecchi. 1983. “Optimization by simulated annealing.” Science 220:671–680.

[5] Carlos M. Fonseca and Peter J. Fleming. “An overview of evolutionary algorithms in multi-objective optimization.” Evolutionary Computation, 3(1):1-16, 1995

[6] D. E. Goldberg and J. Richardson. “Genetic algorithms with sharing for multimodal function optimization.” In Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms, pages 41-49, Hillsdale, N J, 1987. Lawrence Erlbaum

[7] J. David Schaffer. “Multiple objective optimization with vector evaluated genetic algorithms.” In John J. Grefenstette, editor, Proceedings of an International Conference on Genetic Algorithms and Their Applications, pages 93-100, 1985

[8] N. Srinivas and Kalyanmoy Deb. “Multiobjective optimization using nondominated sorting in genetic algorithms.” Evolutionary Computation, 2(3):221-248, 1994

[9] Zitzler E, Thiele L. “Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach.” IEEE Trans Evol Comput 1999;3(4):257-71.

[10] Zitzler, E.; Laumanns, M.; Thiele, L. “SPEA2: Improving the Strength Pareto Evolutionary Algorithm”; Technical Report 103; Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH): Zurich, Switzerland, 2001. Available online at http://www.tik.ee.ethz.ch/sop/publicationListFiles/zlt2001a.pdf.

[11] Knowles, J. D.; Corne, D. W. “The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Pareto Multiobjective Optimisation.” In Proceedings of the 1999 Congress on EVolutionary Computation (CEC1999); IEEE Press: Piscataway, NJ, 1999; pp 98-105.

[12] Martorell S, Carlos S, Sanchez A, Serradell V. “Constrained optimization of test intervals using a steady-state genetic algorithm.” Reliab, Engng Syst Safety 2000;67:215-32

[13] Kaelo P., Ali M.M., “A numerical study of some modified differential evolution algorithm,” European Journal of Operations Research 169, 2006, 1176–1184.
[14] Gujarathi A.M., Babu B.V., “Optimization of Adiabatic Styrene Reactor: A Hybrid Multiobjective Differential Evolution (H-MODE) Approach”, American Chemical Society, 2009
[15] Gujarathi A.M., Sharma D., Babu B.V., “Multi-Objective Optimization of Polyethylene Terephthalate (PET) Reactor using Hybrid Multi-Objective Differential Evolution,” 2007
[16] Parks GT. “Multiobjective pressurized water reactor reload core designusing genetic algorithm search.” Nucl Sci Engng 1997;124:178-87
[17] P. Giuggioli Busacca, M. Marseguerra, E.Zio: “Multiobjective optimization by genetic algorithms: application to safety systems”, Reliability Engineering and System Safety 72, 2001, 59-74
[18] Holland, J. H. (1975). “Adaptation in natural and artificial systems.” Ann Arbor, Michigan: The University of Michigan Press.
[19] Zio E.: “Basics on genetic algorithms with application to system reliability and availability optimization”, Computational Methods For Reliability and Risk Analysis, 2009, 180-186
[20] Radcliff, N. J. 1991. “Forma analysis and random respectful recombination.” In Proc. 4th Int. Conf. on Genetic Algorithms, San Mateo, CA: Morgan Kauffman.
[21] Storn, R., Price, K., “Differential evolution – A simple and efficient adaptive scheme for global optimization over continuous spaces.” Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA. 1995
[22] Storn R., Price K., “Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces”, Journal of Global Optimization 11, 1997, 341–359
[23] Bergey P.K., Ragsdale C. “Modified differential evolution: a greedy random strategy for genetic recombination.” The International Journal of Management Science, 2004
[24] Hui-Yuan Fan, Jouni Lampinen, “A Trigonometric Mutation Operation to Differential Evolution”, Journal of Global optimization 2003, 105-129.
[25] Pant M., Ali M.M., Singh V.P., “Differential evolution with Parent Centric Crossover”, Second UKSIM European Symposium on Computer Modeling and Simulation, 2008
[26] Ali M.M.: “Differential evolution with preferential crossover.” European Journal of Operations Research 181, 2007, 1088–1113

[27] Pant M., Ali M.M., Singh V.P., “Two modified differential evolution algorithms and their applications to engineering design problems”, World Journal of Modelling and Simulation, 2010, 72-80
[28] Yang Z., He J., Yao X., “Making a difference to Difference Evolution”, Advances in Metaheuristics for Hard Optimization, 2007, 397-413
[29] Das S. et al.: “Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives”, Studies in Computational Intelligence (SCI) 116, 1–38 (2008)
[30] Rahnamayan S., Tizhoosh H.R., Salama M.M.A.: Opposition-Based Differential Evolution Algorithms, Advances in Differential Evolution, 2008, 155-171
[31] Rahnamayan S., Tizhoosh H.R., Salama M.M.A: Opposition versus randomness in soft computing techniques, Applied Soft computing 8, 2008, 906-918
[32] Ventresca M., Rahnamayan S., Tizhoosh H.R.: A note on “Opposition versus randomness in soft computing techniques, Applied Soft computing 8”, Applied Soft Computing 10, 2010, 956-957
[33] Babu, B. V. and Angira, R., “Modified Differential Evolution (MDE) for Optimization of Non-Linear Chemical Processes”, Computers & Chemical Engineering 30, 2006, 989–1002.
[34] Ali M.M., Pant M., Abraham A., “A Modified Differential Evolution Algorithm and Its Application to Engineering Problems”, International Conference of Soft Computing and Pattern Recognition, 2009
[35] Babu B. V and Angira R, “Optimization Using Hybrid Differential Evolution Algorithms”, ChemCon 2004, Mumbai
[36] Pant M., Ali M.M., Abraham A, “Hybrid Differential Evolution – Particle Swarm Optimization Algorithm for Solving Global Optimization Problems”, 2008
[37] Michalewicz, Z., Schoenauer, M.: “Evolutionary Algorithms for Constrained Parameter Optimization Problems.” Evolutionary Computation 4, 1996, 1–32
[38] J. Lampinen, I.Zelinka, “On stagnation of the differential evolution algorithm”
[39] R. Storn, “Differential Evolution Research – Trends and Open Questions”, Advances in Differential Evolution, 2008, 1-31
[40] D. Zaharie, “A Comparative Analysis of Crossover Variants in Differential Evolution”, Proceeding of IMCSIT 2007, U. Markowaska-Kaczmar and H. Kwasnicka, Eds. Wisla: PTI 2007, 171-181
[41] D. Zaharie, “Critical values for the control parameters of differential evolution algorithms.” 2002a In: Matousek, R., Osmera, P. (Eds.), In: Proceedings of the 8th International Conference on Soft Computing, Brno, pp. 62–67.

[42] M. M. Ali and A. Törn, “Population set-based global optimization algorithms: Some modifications and numerical studies,” Comput. Oper. Res., vol. 31, no. 10, pp. 1703–1725, 2004.
[43] Zaharie, D., 2003. “Control of population diversity and adaptation in differential evolution algorithms.” In: Matousek, R., Osmera, P. (Eds.), In: Proceedings of the 9th International Conference on Soft Computing, Brno, pp. 41–46.
[44] J. Liu and J. Lampinen, “A fuzzy adaptive differential evolution algorithm”, Soft Computing—A Fusion of Foundations, Methodologies and Applications, vol. 9, no. 6, pp. 448–462, 2005 [Online].
[45] J. Tvrdík, “Competitive differential evolution,” in MENDEL 2006, 12th International Conference on Soft Computing, R. Matoušek and P. Ošmera, Eds. Brno: University of Technology, 2006, pp. 7–12.
[46] L. Wu, Y. Wang, S. Zhou, “Self-Adapting Control Parameters Modified Differential Evolution for Trajectory Planning of Manipulators”, Journal of Control Theory and Applications, 2007, 365-373.
[47] J. Brest, S. Greiner, B. Boškovič, M. Mernik, and V. Žumer, “Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems,” IEEE Transactions on Evolutionary Computation, vol. 10, pp. 646–657, 2006.
[48] J. Brestm, V. Žumer, M.S. Maučec, “Self-Adaptive Differential Evolution Algorithm in Constrained Real-Parameter Optimization”, IEEE Congress on Evolutionary Computation, 2006.
[49] A. K. Qin and P. N. Suganthan. “Self-adaptive Differential Evolution Algorithm for Numerical Optimization”. In The 2005 IEEE Congress on Evolutionary Computation CEC2005, volume 2, pages 1785–1791.
[50] A. Salman, A.P. Engelbrecht, M.G.H. Omran, “Empitical Analysis of Self-Adaptive Differential Evolution”, European Journal of Operational Research, 2007, 785-804.
[51] Abbass H.A., “The Self-Adaptive Pareto Differential Evolution Algorithm”
[52] Gao, X.; Chen, B.; He, B.; Qiu, T.; Li, J.; Wang, C.; Zhang, L. “Multi-objective optimization for the periodic operation of the naphtha pyrolysis process using a new parallel hybrid algorithm combining NSGA-II with SQP.” Comput. Chem. Eng. 2008, 32, 2801.
[53] Agarwal, A.; Gupta, S. K. “Jumping gene adaptations of NSGA-II and their use in the multi-objective optimal design of shell and tube heat exchangers.” Chem. Eng. Res. Des. 2007, 86, 123.
[54] Abbass H.A., Sarker R., Newton C., PDE: “A Pareto-frontier Differential Evolution Approach for Multi-objective Optimization Problems”, 2001
[55] Abbass H.A., “The Self-Adaptive Pareto Differential Evolution Algorithm”

[56] Babu, B. V.; Chakole, P. G.; Mubeen, J. H. S. 2Multi-objective Differential Evolution (MODE) for Optimization of Adiabatic Styrene Reactor." Chem. Eng. Sci. 2005, 60, 4822.
[57] Babu, B. V.; Gujarathi, A. M.; Katla, P.; Laxmi, V. B. Strategies of Multi-Objective Differential Evolution (MODE) for Optimization of Adiabatic Styrene Reactor. In Proceedings of the International Conference on Emerging Mechanical Technology: Macro to Nano (EMTMN-2007); p 243.
[58] Gujarathi A.M., Lohumi A., Mishra M., Sharma D., Babu B.V., "Multi-Objective Optimization using Trigonometric Mutation Multi-Objective Differential Evolution Algorithm", 2009
[59] M. Montaz Ali, Charoenchai Khompatraporn, Zelda B. Zabinsky, "A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems", Journal of Global Optimization 31, 2005, 635-672
[60] E. Zitzler, K. Deb, L. Thiele, "Comparison of Multi-objective Evolutionary Algorithms: Empirical Results", 1999
[61] Yang J-E, Hwang M-J, Sung T-Y, Jin Y. "Application of genetic algorithm for reliability allocation in nuclear power plants". Reliab Engng Syst Safety 1999;65:229

Appendix A:

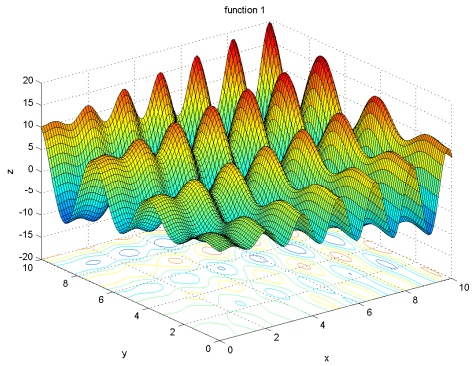
Benchmark problems for single-objective optimization

The appendix presents 23 benchmark functions, with their domain, minimum's value and location in the search space.

f1. Example function taken from: Practical Genetic Algorithms, second edition, John Wiley & Sons

$$\min_{\underline{x}} f(\underline{x}) = x_1 \sin(4x_1) + 1.1 \cdot x_2 \sin(2x_2) \qquad 0 \le x_i \le 10$$

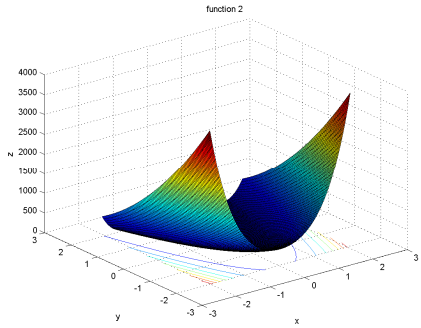
$$f(\underline{x}^*) = -18.5547 \quad \underline{x}^* = (9.039, 8.668)$$



f2. Second De Jong function, Rosenbrock's saddle

$$\min_{\underline{x}} f(\underline{x}) = \sum_{i=1}^{n-1} \left[100 \cdot \left(x_{i+1} - x_i^2 \right)^2 + (1 - x_i)^2 \right] \qquad -2.048 \le x_i \le 2.048$$

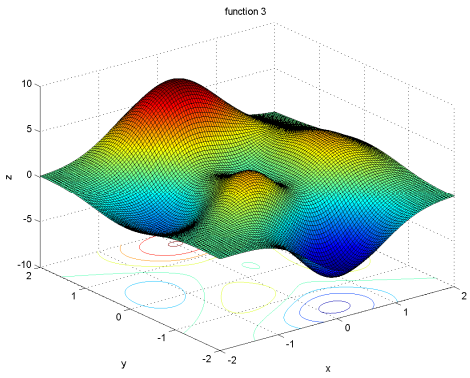
$$f(\underline{x}^*) = 0 \quad \underline{x}^* = (1,1,\ldots,1)$$



f3. Peaks function from Matlab

$$\min_{\underline{x}} peaks(\underline{x}) \qquad -2 \le x_i \le 2$$

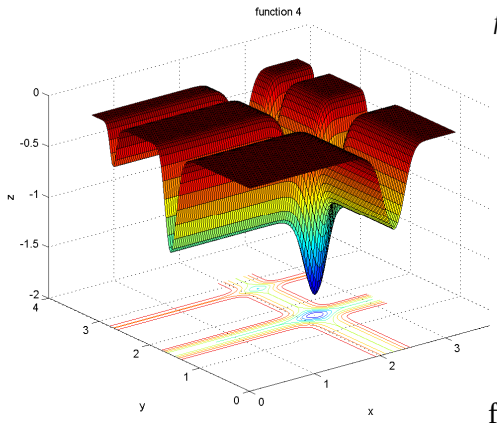
$$f(\underline{x}^*) = -6.5511 \quad \underline{x}^* = (0.2283, -1.6255)$$



f4. Michalewicz function

$$\min_{\underline{x}} f(\underline{x}) = -\sum_{i=1}^{n} \sin(x_i) \cdot \sin\left(i \frac{x_i^2}{\pi}\right)^{2m} \qquad 0 \le x_i \le \pi \quad m = 10$$

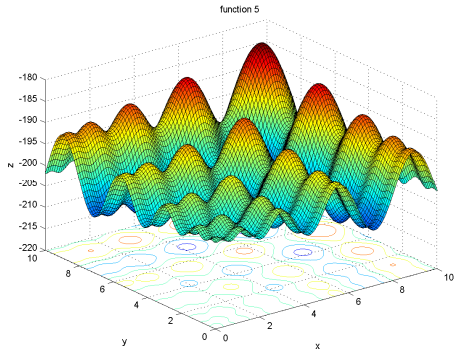
$$for\ n = 2 \quad f(\underline{x}^*) = -1.8013 \quad \underline{x}^* = (2.2029, 1.5708)$$



f5. Schwefel's problem

$$\min_{\underline{x}} f(\underline{x}^*) = -100 \cdot n - \sum_{i=1}^{n} x_i \sin\left(\sqrt{|100 \cdot x_i|}\right) \qquad 0 \le x_i \le 10$$

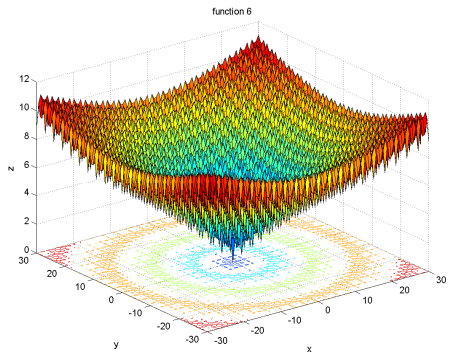
$$for\ n = 4 \quad f(\underline{x}^*) = -428.6030 \quad \underline{x}^* = (7.1706, 7.1706, 7.1706, 7.1706)$$



f6. Ackley's problem, Storn and Price, 1997

$$\min_{\underline{x}} f(\underline{x}) = -20 \cdot e^{\left(-0.02\sqrt{n^{-1}\sum_{i=1}^{n} x_i^2}\right)} - e^{\left(n^{-1}\sum_{i=1}^{n} \cos(2\pi x_i)\right)} + 20 + e \qquad -30 \le x_i \le 30$$

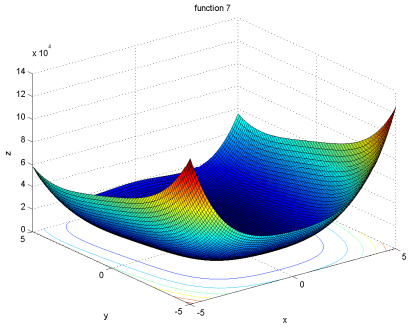
$$f(\underline{x}^*) = 0 \quad \underline{x}^* = (0, 0, \ldots, 0)$$



f7. Modified Rosenbrock problem, Price, 1977

$$\min_{\underline{x}} f(\underline{x}) = 100\left(x_2 - x_1^2\right)^2 + \left[6.4\left(x_2 - 0.5\right)^2 - x_1 - 0.6\right]^2 \qquad -5 \le x_i \le 5$$

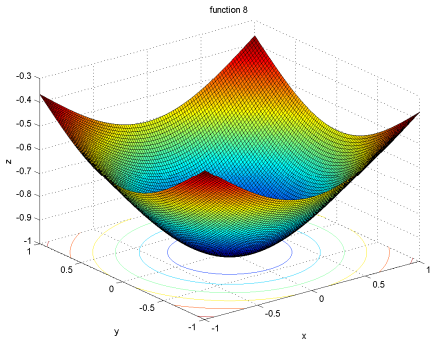
$$f(\underline{x}^*) = 0 \quad \underline{x}^* \approx (0.3412, 0.1164) \text{ and } \underline{x}^* \approx (1, 1)$$



f8. Exponential problem, Breiman and Cutler, 1993

$$\min_{\underline{x}} f(\underline{x}) = -e^{\left(-0.5\sum_{i=1}^{n} x_i^2\right)} \qquad -1 \le x_i \le 1$$

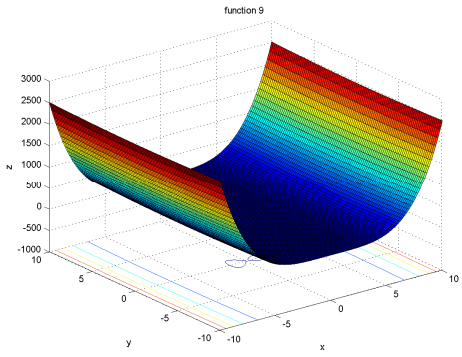
$$f(\underline{x}^*) = -1 \quad \underline{x}^* = (0, 0, \ldots, 0)$$



f9. Aluffi-Pentini's problem, 1985

$$\min_{\underline{x}} f(\underline{x}) = 0.25x_1^4 - 0.5x_1^2 + 0.1x_1 + 0.5x_2^2 \qquad -10 \le x_i \le 10$$

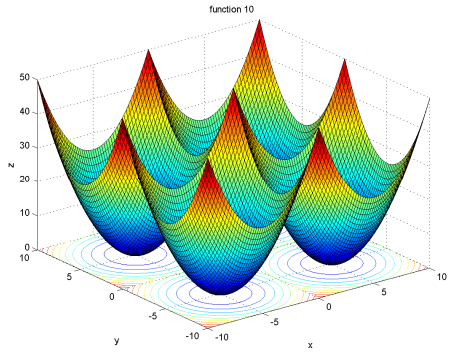
$$f(\underline{x}^*) \approx -0.3523 \quad \underline{x}^* = (-1.0465, 0)$$



f10. Becker and Lago problem, Price 1977

$$\min_{\underline{x}} f(\underline{x}) = \left(|x_1| - 5\right)^2 + \left(|x_2| - 5\right)^2 \qquad -10 \le x_i \le 10$$

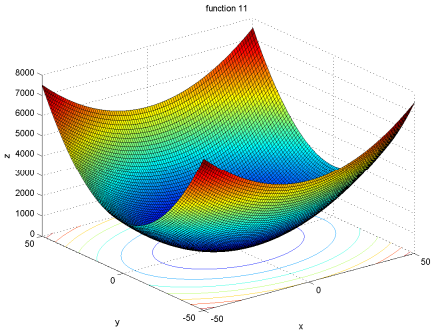
$$f(\underline{x}^*) = 0 \qquad \underline{x}^* = (\pm 5, \pm 5)$$



f11. Bohachevsky problem 1, 1986

$$\min_{\underline{x}} f(\underline{x}) = x_1 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7 \qquad -50 \le x_i \le 50$$

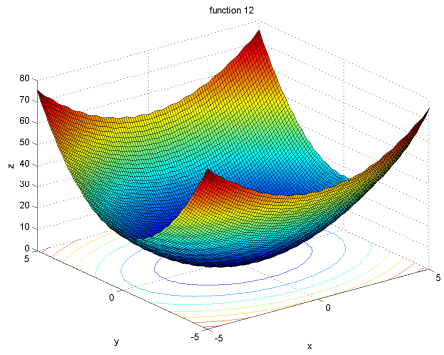
$$f(\underline{x}^*) = 0 \quad \underline{x}^* = (0,0)$$



f12. Bohachevsky problem 1, 1986

$$\min_{\underline{x}} f(\underline{x}) = x_1 + 2x_2^2 - 0.3\cos(3\pi x_1) \cdot 0.4\cos(4\pi x_2) + 0.7 \qquad -50 \le x_i \le 50$$

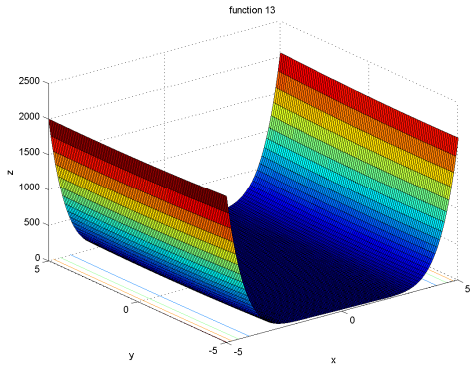
$$f(\underline{x}^*) = 0 \quad \underline{x}^* = (0,0)$$



f13. Camel back – 3, Three hump problem, Dixon and Szegö, 1975

$$\min_{\underline{x}} f(\underline{x}) = 2x_1^2 - 1.05x_1^4 + \frac{1}{6}x_1^6 + x_1 \cdot x_2 + x_2^2 \qquad -5 \le x_i \le 5$$

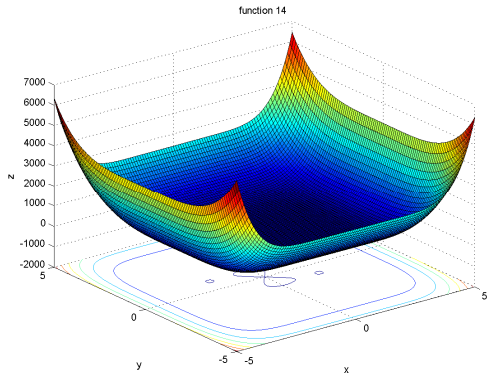
$$f(\underline{x}^*) = 0 \quad \underline{x}^* = (0,0)$$



f14. Camel back – 6, Six hump problem, Dixon and Szegö, 1978

$$\min_{\underline{x}} f(\underline{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 \cdot x_2 - 4x_2^2 + 4x_2^4 \qquad -5 \le x_i \le 5$$

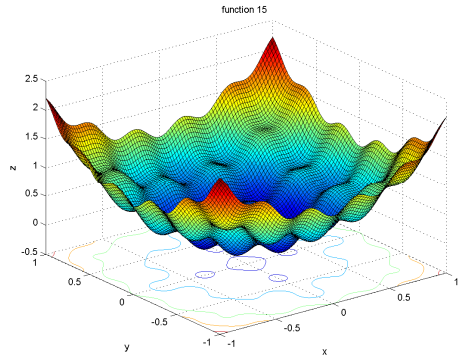
$$f(\underline{x}^*) \approx -1.0316 \quad \underline{x}^* \approx (0.089842, -0.712656) \text{ and } \underline{x}^* \approx (-0.089842, 0.712656)$$



f15. Cosine mixture problem, Breiman and Cutler, 1993

$$\min_{\underline{x}} f(\underline{x}) = \sum_{i=1}^{n} x_i^2 - 0.1\sum_{i=1}^{n}\cos(5\pi x_i) \qquad -1 \le x_i \le 1$$

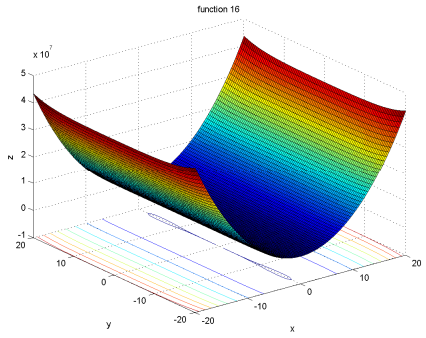
$$f(\underline{x}^*) = -0.1n \quad \underline{x}^* = (0,0,\ldots,0)$$



f16. Dekkers and Aarts problem, 1991

$$\min_{\underline{x}} f(\underline{x}) = 10^5 x_1^2 + x_2^2 - \left(x_1^2 + x_2^2\right)^2 + 10^{-5}\left(x_1^2 + x_2^2\right)^4 \qquad -20 \le x_i \le 20$$

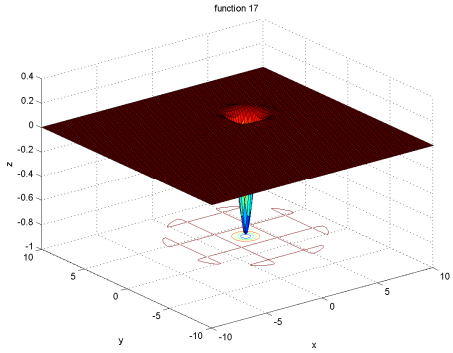
$$f(\underline{x}^*) = -24777.4817 \quad \underline{x}^* = (0,15) \text{ and } \underline{x}^* = (0,-15)$$



f17. Eason problem, Michalewicz, 1996

$$\min_{\underline{x}} f(\underline{x}) = -\cos(x_1)\cdot\cos(x_2)\cdot e^{\left(-(x_1-\pi)^2-(x_2-\pi)^2\right)}$$

$$f(\underline{x}^*) = -1 \quad \underline{x}^* = (\pi,\pi) \qquad -10 \le x_i \le 10$$

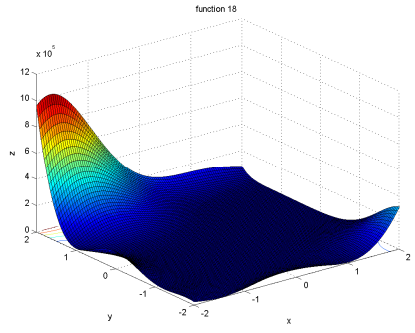


f18. Goldstein and Price, Dixon and Szegö, 1978

$$\min_{\underline{x}} f(\underline{x}) = \left[1+(x_1+x_2+1)^2\cdot(19-14x_1+3x_1^2-14x_2+6x_1\cdot x_2+3x_2^2)\right]\cdot$$

$$\cdot\left[30+(2x_1-3x_2)^2\cdot(18-32x_1+12x_1^2+48x_2-36x_1\cdot x_2+27x_2^2)\right] \qquad -2 \le x_i \le 2$$

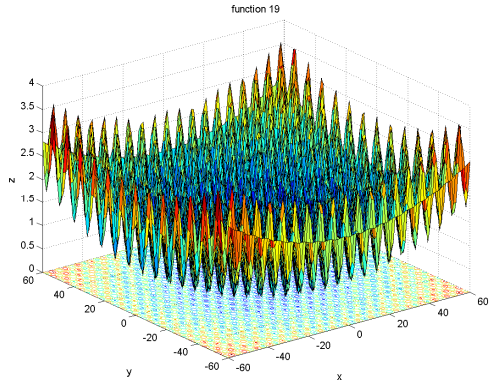
$$f(\underline{x}^*) = 3 \quad \underline{x}^* = (0,-1)$$



f19. Griewank problem, 1981

$$\min_{\underline{x}} f(\underline{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$$

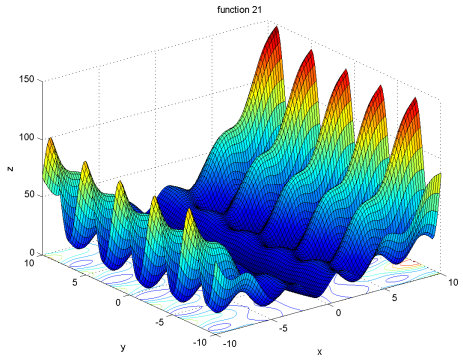
$$f(\underline{x}^*) = 0 \quad \underline{x}^* = (0,0,\ldots,0) \quad -600 \le x_i \le 60$$



$$\min_{\underline{x}} f(\underline{x}) = \left(\frac{\pi}{n}\right)\cdot\left(10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i-1)^2\cdot\left[1+10\sin^2(\pi y_{i+1})\right] + (y_{n-1}-1)^2\right)$$

$$y_i = 1 + \frac{1}{4}(x_i+1) \qquad -10 \le x_i \le 10$$

$$f(\underline{x}^*) = 0 \quad \underline{x}^* = (-1,-1,\ldots,-1)$$



f20. Helical valley problem, Wolfe, 1978

$$\min_{\underline{x}} f(\underline{x}) = 100\left[(x_2 - 10\cdot\theta)^2 + \left(\sqrt{(x_1^2+x_2^2)}-1\right)^2\right] + x_3^2$$

$$\theta = \begin{cases} \frac{1}{2\pi}\arctan\left(\frac{x_2}{x_1}\right) & \text{if } x_1 \ge 0 \\ \frac{1}{2\pi}\arctan\left(\frac{x_2}{x_1}\right) + \frac{1}{2} & \text{if } x_1 < 0 \end{cases} \qquad -10 \le x_i \le 10$$

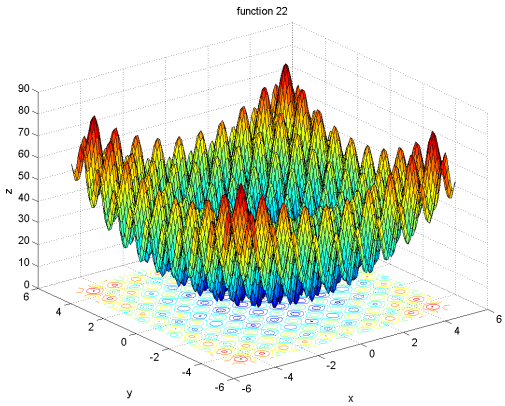
$$f(\underline{x}^*) = 0 \quad \underline{x}^* = (1,0,0)$$

f21. Levy and Montalvo problem 1, 1985

f22. Rastrigin problem, Storn and Price, 1997

$$\min_{\underline{x}} f(\underline{x}) = 10n + \sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i)\right] \qquad -5.12 \le x_i \le 5.12$$

$$f(\underline{x}^*) = 0 \quad \underline{x}^* = (0,0,\ldots,0)$$

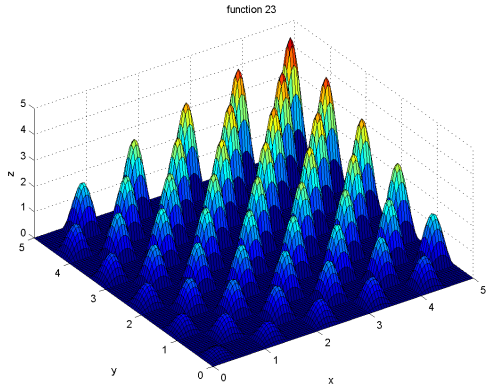


f23. Function taken from E. Zio lectures

$$y(\underline{x}) = -\sqrt{x_1}\sin(2\pi x_1)\sqrt{x_2}\sin(2\pi x_2)$$

$$\min_{\underline{x}} f(\underline{x}) = \begin{cases} y(\underline{x}) & \text{if } y(\underline{x}) < 0 \\ 0 & \text{otherwise} \end{cases} \qquad 0 \le x_i \le 5$$

$$f(\underline{x}^*) \approx -4.7513 \quad \underline{x}^* \approx (4.7527, 4.7527)$$



Appendix B

Benchmark problems for multi-objective optimization

The appendix presents 3 benchmark problems with high dimensionality for multi-objective optimization, with domains and location of the Pareto optimal frontiers.

General task: $$\min_{\underline{x}} \underline{F}(\underline{x}) = \left(f_1(x_1), f_2(\underline{x}) \right) \tag{B.1}$$

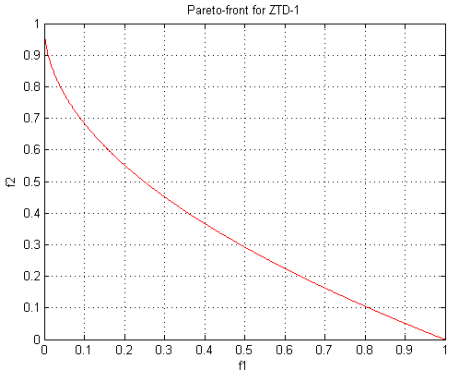
F1. This test is known also as ZTD1; The Pareto optimal front is formed with $g(\underline{x})=1$

$$f_1(x_1) = x_1$$

$$g(x_2,\ldots,x_n) = 1 + 9 \cdot \sum_{i=2}^{n} x_i$$

$$f_2(f_1, g) = 1 - \sqrt{\frac{f_1}{g}}$$

$$n = 30, \quad 0 \le x_i \le 1$$



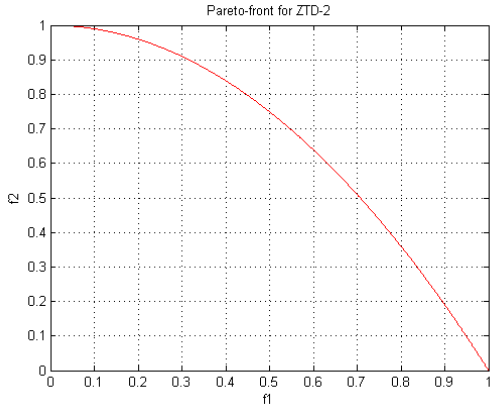
F2. This test is known also as ZTD2; The Pareto optimal frontier is formed when $g(\underline{x})=1$

$$f_1(x_1) = x_1$$

$$g(x_2,\ldots,x_n) = 1 + 9 \cdot \sum_{i=2}^{n} x_i$$

$$f_2(f_1, g) = 1 - \left(\frac{f_1}{g} \right)^2$$

$$n = 30, \quad 0 \le x_i \le 1$$



F3. This test is known also as ZTD3, the red line is the objectives' search space for the best Pareto frontier showed by the green markers; The Pareto optimal front is formed with $g(\underline{x})=1$

$$f_1(x_1) = x_1$$

$$g(x_2,\ldots,x_n) = 1+9\cdot\sum_{i=2}^{n} x_i$$

$$f_2(f_1,g) = 1-\sqrt{\frac{f_1}{g}}-\sin(10\pi f_1)$$

$$n=30,\quad 0\le x_i \le 1$$