

Neoteric Differential Evolution

In this chapter you will make the acquaintance of the newest statement of differential evolution. But first, I propose that you dip into the background of population-based methods, so-called evolutionary algorithms. Also, I shall show the basic evolutionary algorithm scheme and apply it to differential evolution. After a rigorous mathematical definition of the optimization problem, I shall show you the fresh wording of the differential evolution algorithm, then together we shall compare it with the classical one, described in Chapter 1, and emphasize its advantages. In conclusion I shall point out the efficient techniques to handle mixed variables as well as constraints.

2.1 Evolutionary Algorithms

Let us talk a little more about Evolutionary Algorithms (EA). As you have already heard, these are population-based metaheuristics that can be applied with success in cases where traditional methods do not give satisfactory results. Originally inspired by the theory of evolution proposed by Darwin, these methods gave birth to the whole discipline, Evolutionary computation [SDB⁺93], that involves the simulation of natural evolution processes on a computer.

Evolutionary algorithms emerged in the sixties. Initially, EA were presented by three general trends. These are the genetic algorithms, evolution strategy, and evolutionary programming. Later, in the early nineties, the fourth trend, genetic programming, has come to light.

Genetic Algorithms. This is one of the most popular ideas in evolutionary computation. The concept of genetic algorithms was introduced and developed by J. Holland [Hol75]. In order to achieve a better understanding of the biological adaptation mechanisms he tried to simulate these processes numerically. That, in turn, resulted in the first genetic algorithm. Soon afterwards,

K. DeJong [DeJ75] formalized genetic algorithms for the binary search space. And some years later, thanks to D. Goldberg [Gol89], genetic algorithms became widely available.

Evolution Strategies were proposed by I. Rechenberg [Rec73] and H. Schwefel [Sch81]. Solving aviation engineering problems, for which classical optimization suffers a defeat, they revealed the most important key positions in evolutionary algorithms, namely, the ideas of adaptation and self-adaptation for control parameters of an algorithm.

Evolutionary Programming was elaborated by L.J. Fogel [FOW66]. Working on the evolution of finite state machines to predict time series, he gave birth to a new evolutionary branch. Being the result of, or to be more exact, the desire to procreate machine intelligence, evolutionary programming finally became an efficient optimizer. Later, this trend was appreciably enlarged by D.B. Fogel [Fog92].

Genetic Programming, successfully introduced by J. Koza [Koz92], arose from the evolution of more complex structures such as a set of expressions of a programming language and neural networks. J. Koza presented the structure (individual) in the form of trees, orientable graphs without cycles, in which each of the nodes is associated with a unit operation related to the problem domain.

For a deeper examination of this topic I definitely suggest a quite recent reference book of A.E. Eiben and J.E. Smith [ES03]. And now we shall consider the basic scheme that generalizes all evolutionary algorithms (see Fig. 2.1).

Evolutionary Algorithms. The vocabulary of evolutionary algorithms is, to a great extent, borrowed from both the biology and the theory of evolution. A set of problem parameters, *genes*, is described by an *individual*. An ensemble of individuals composes a *population*. Before optimizing, EA is initialized, often randomly because usually we do not have ideas about optimum localization, by a population of individuals. We name this *initialization*. Next, the optimization criterion, in the EA case so-called *fitness*, is calculated for each individual of the population. This is *evaluation*. Sometimes evaluation of fitness can be a computationally intensive operation. Thus, the initial population of *Parents* is ready and the algorithm begins its *evolutionary cycle*. Iterations, in EA terms, *generations*, last until a *stopping condition* is attained. In each evolutionary cycle the population passes through the following three steps.

1. *Selection* of the individuals that are more apt to reproduce themselves, from the population.
2. *Variations* of the selected individuals in a random manner. Mainly two operations are distinguished here: *crossover* and *mutation*. The variations of Parents germinate *Children*.
3. *Replacement* refreshes the population of the next generation usually by the best individuals chosen among Parents and Children.

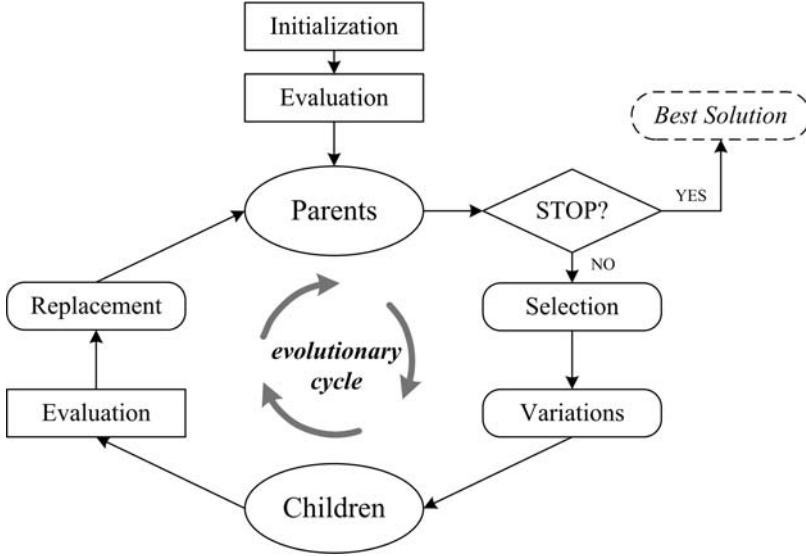


Fig. 2.1. A basic scheme of evolutionary algorithms.

Evolutionary Algorithm typically can be outlined by the following way (see Alg. 2).

Algorithm 2 Typical Evolutionary Algorithm

```

generation  $g \leftarrow 0$ 
population  $\mathbb{P}^g \leftarrow \text{Initialize}$ 
fitness  $f(\mathbb{P}^g) \leftarrow \text{Evaluate}$ 
while (not stopping condition) do
  // proceed to the next evolutionary cycle //
   $g \leftarrow g + 1$ 
  Parents  $\leftarrow \text{Select from } \mathbb{P}^g$ 
  Children  $\leftarrow \text{Vary Parents (Crossover, Mutation, ...)}$ 
  fitness  $\leftarrow \text{Evaluate Children}$ 
  Replacement  $\mathbb{P}^g \leftarrow \text{Survive Parents and Children}$ 
end while

```

2.2 Problem Definition

The overall goal of an optimization problem $f : M \subseteq \mathbb{R}^D \rightarrow \mathbb{R}, M \neq \emptyset$, where f is called the *objective function* (also *fitness* or *cost function*), is to find a vector $X^* \in M$ such that:

$$\forall X \in M : f(X) \geq f(X^*) = f^*, \quad (2.1)$$

where f^* is called a *global minimum*; X^* is the *minimum location* (*point* or *set*).

$$M = \{X \in \mathbb{R}^D \mid g_k(X) \leq 0, \forall k \in \{1, \dots, m\}\} \quad (2.2)$$

is the set of feasible points for a problem with inequality constraints $g_k : \mathbb{R}^D \rightarrow \mathbb{R}$.

A particular case of inequality constraints is boundary constraints

$$L \leq X \leq H : \quad L, H \in \mathbb{R}^D. \quad (2.3)$$

For an unconstrained problem $M = \mathbb{R}^D$.

Because $\max\{f(X)\} = -\min\{-f(X)\}$, the restriction to minimization is without loss of generality. In general the optimization task is complicated by the existence of nonlinear objective functions with multiple local optima. A *local minimum* $\hat{f} = f(\hat{X})$ is defined by the condition (2.4).

$$\exists \epsilon : \forall X \in M \mid \|X - \hat{X}\| < \epsilon \Rightarrow \hat{f} \leq f(X). \quad (2.4)$$

2.3 Neoteric Differential Evolution

As with all evolutionary algorithms, differential evolution deals with a population of solutions. The population $\mathbb{P}$ of a generation g has NP vectors, so-called individuals of population. Each such individual represents a potential optimal solution.¹

$$\mathbb{P}^g = \{X_i^g\}, \quad i = 1, \dots, NP. \quad (2.5)$$

In turn, the individual contains D variables, so-called *genes*.

$$X_i^g = \{x_{i,j}^g\}, \quad j = 1, \dots, D. \quad (2.6)$$

Usually, the population is initialized by randomly generating individuals within the boundary constraints (2.3),

$$\mathbb{P}^0 = \{x_{i,j}^0\} = \{rand_{i,j} \cdot (h_j - l_j) + l_j\}, \quad (2.7)$$

where the *rand* function uniformly generates values in the interval $[0, 1]$.

¹ In order to show the flexibility of implementation, here I represent a population and an individual as a set of elements instead of a vector presentation.

Then, for each generation all the individuals of the population are updated by means of a reproduction scheme. Thereto for each individual *ind* a set π of other individuals is randomly extracted from the population. To produce a new one the operations of differentiation and crossover are applied one after another. Next, selection is used to choose the best. Let us consider these operations in detail.

First, a set of randomly extracted individuals $\pi = \{\xi_1, \xi_2, \dots, \xi_n\}$ is necessary for differentiation. The strategies (i.e., a difference vector δ and a base vector β) are designed on the basis of these individuals. Thus, the result of differentiation, the so-called *trial* individual, is

$$\omega = \beta + F \cdot \delta, \quad (2.8)$$

where F is the constant of differentiation. I shall show an example of a typical strategy [SP95]. Three different individuals are randomly extracted from the population. The trial individual is equal to $\omega = \xi_3 + F \cdot (\xi_2 - \xi_1)$ with the difference vector $\delta = \xi_2 - \xi_1$ and the base vector $\beta = \xi_3$.

Afterwards, the trial individual ω is recombined with the target one *ind*. Crossover represents a typical case of a gene's exchange. A new trial individual inherits genes of the target one with some probability. Thus,

$$\omega_j = \begin{cases} \omega_j & \text{if } rand_j \geq Cr \\ ind_j & \text{otherwise} \end{cases} \quad (2.9)$$

where $j = 1, \dots, D$, $rand_j \in [0, 1)$ and $Cr \in [0, 1]$ is the constant of crossover. This was a combinatorial crossover. Also, other types of crossover can be used: binary approach [SP95], mean-centric (UNDX, SPX, BLX), and parent-centric (SBX, PCX) approaches [DJA01].

Selection is realized by simply comparing the objective function values of target and trial individuals. If the trial individual better minimizes the objective function, then it replaces the target one. This is the case of *elitist* or so-called “greedy” selection.

$$ind = \begin{cases} \omega & \text{if } f(\omega) \leq f(ind) \\ ind & \text{otherwise.} \end{cases} \quad (2.10)$$

Notice that there are only three control parameters in this algorithm. These are:

- NP – population size
- F – constant of differentiation
- Cr – constant of crossover

As for stopping conditions, one can either fix the number of generations $g_{\max}$ or a desirable precision of the solution *VTR* (*value-to-reach*).

The pattern of the DE algorithm is presented hereafter (see Alg. 3).

Algorithm 3 Neoteric Differential Evolution

Require: F, Cr, NP – control parameters
 initialize $\mathbb{P}^0 \leftarrow \{ind_1, \dots, ind_{NP}\}$
 evaluate $f(\mathbb{P}^0) \leftarrow \{f(ind_1), \dots, f(ind_{NP})\}$
while (**not** stopping condition) **do**
 for all $ind \in \mathbb{P}^g$ **do**
 $\mathbb{P}^g \rightarrow \pi = \{\xi_1, \xi_2, \dots, \xi_n\}$
 $\omega \leftarrow \text{Differentiation}(\pi, F, \text{Strategy})$
 $\omega \leftarrow \text{Crossover}(\omega, Cr)$
 $ind \leftarrow \text{Selection}(\omega, ind)$
end for
 $g \leftarrow g + 1$
end while

2.4 Distinctions and Advantages

Above all, I would like to compare differential evolution to the basic EA scheme. As you have already observed, initialization and evaluation are kept without changes. A general EA provides for Darwin’s mechanism of parent selection, however, where more apt it stands a better chance to reproduce itself; differential evolution applies variations (differentiation and crossover) sequentially to each individual. For that, an ensemble of individuals is randomly chosen from the population each time. The result of variations is child, called a trial individual. Moreover, in DE the trial immediately replaces its ancestor in the population if its fitness is better than or equal to its ancestor’s. Also, a stopping condition is verified right here after replacement. Finally, for more clarification I propose that you familiarize yourself with the individual’s cycle in differential evolution, which I presented in Fig.2.2.

Furthermore, let us touch nicety and compare neoteric differential evolution with the classical one, described in Chapter 1.

The first point that is significant is the dissociation of differentiation and crossover in the new DE statement. Of course, we can always associate back these two operations as soon as we need them. However, such a dissociation offers us self-evident advantages. This is, at the minimum, the independent study and use of the operations that enables us to exclude one operation from another and thoroughly analyze their behavior and influence on the search process. Next, differential mutation of classic DE is generalized to differentiation of a neoteric one. From a theoretical point of view, it gives the unlimited spectrum of strategies that obey the unique and universal principle of optimization $\omega = \beta + F \cdot \delta$. In practice, we can now manipulate with great ease the search strategies according to the needs of an optimization task.

The second significant point is the crossover operation in itself. Now the basic element of crossover is a trial individual created in the issue of differentiation, rather than a current one. This improvement changes the philosophy

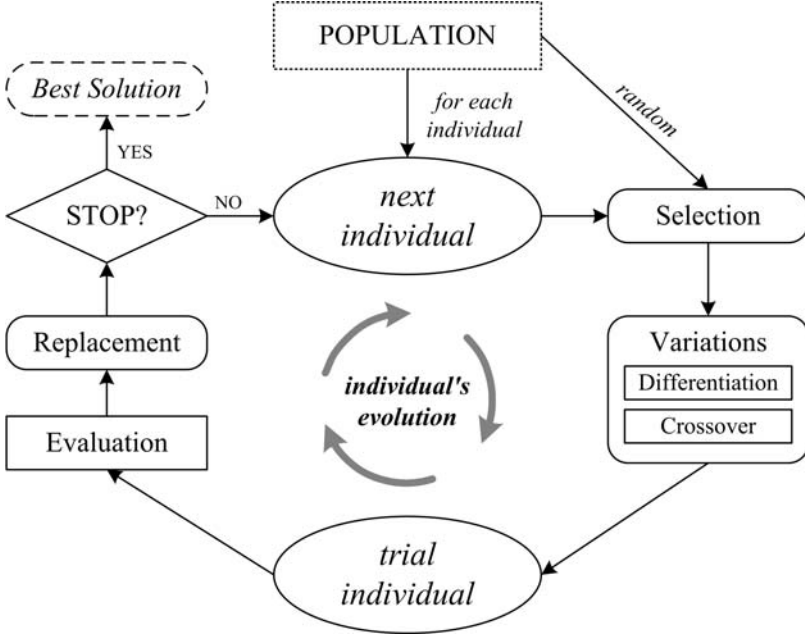


Fig. 2.2. The evolutionary cycle of an individual in differential evolution.

of the solution search. If before the variation operations are considered as the mutation of a current individual resembling evolutionary strategies, then now the main attention is completely focused on the creation of a new, more perfect individual. Such an individual is produced primarily on the basis of the actual state of the population and certainly may inherit some properties of a current individual.

Also, I moved away the mutation parameter $Rnd \in [1, \dots, D]$ (see (1.2)). I do not consider it very important for optimization. Besides, we can almost always imitate it by the appropriate choice of the crossover value Cr . For example, following (2.9), $Cr = 0$ (absence of crossover) $\Rightarrow$ the new-created individual is completely inherited; $Cr = 1 \Rightarrow$ the current individual is completely inherited; and $Cr \approx 1 - 1/D$ permits us to inherit the minimal number of the new-created individual's genes. Although it does not guarantee absolutely that at least one new gene passes into the next generation (the case of classical DE), it certainly guarantees that there is a great chance it does happen.

Here I emphasized three **principal advantages** of the new algorithm's statement. Perhaps, in future, continuously working with differential evolution you will find many more advantages than I did. So, these are as follows.

1. *Efficiency.* A special stress is laid on the efficient creation of a new member of a population, instead of the mutation of current individuals.

2. *Flexibility.* The new algorithm is more flexible to use and adapts to modification; it is preferred for research purposes. In particular, the isolation in the reproduction cycle of differentiation, crossover, and selection from one another in action allows natural and easy driving by the evolution process.
3. *Fundamentality.* The well-known algorithm stated in Chapter 1 is just a particular case of neoteric differential evolution. In fixing the differentiation strategy and appropriate crossover, variation operations can be convoluted to a single equation similar to (1.2). Moreover, differentiation synthesizes in itself the fundamental ideas in optimization.² The operation of Differentiation intrinsically generalizes the universal concepts of the solution search, as in the case of traditional versus modern methods of optimization.

2.5 Mixed Variables

Differential evolution in its initial form is a method for continuous variable optimization [SP95]. However, in [LZ99b, LZ99c] the DE modification for integer and discrete variables is proposed. We first show **integer variable** handling.

Despite the fact that DE works with continuous values on the bottom level, for the evaluation of the objective function integer values are used. Thus,

$$\begin{aligned}
 f &= f(Y) : Y = \{y_i\} \\
 \text{where} \quad y_i &= \begin{cases} x_i & \text{for continuous variables} \\ \lfloor x_i \rfloor & \text{for integer variables} \end{cases}, \\
 \text{and} \quad X &= \{x_i\}, \quad i = 1, \dots, D.
 \end{aligned} \tag{2.11}$$

The $\lfloor x \rfloor$ function gives the nearest integer less than or equal to x . Such an approach provides a great variety of individuals and ensures algorithm robustness. In other words, there is no influence from discrete variables on algorithm functioning. In the integer case, the population initialization occurs as follows.

$$\mathbb{P}^0 = \{x_{i,j}^0\} = \{rand_{i,j} \cdot (h_j - l_j + 1) + l_j\}, \quad rand_{i,j} \in [0, 1). \tag{2.12}$$

Next, **discrete variables** can be designed in the same easy way. It is supposed that a discrete variable $Z^{(d)}$ takes its values from the discrete set $\{z_i^{(d)}\}$ containing l ordered elements.

$$\begin{aligned}
 Z^{(d)} &= \{z_i^{(d)}\}, \quad i = 1, \dots, l \\
 \text{so that} \quad z_i^{(d)} &< z_{i+1}^{(d)}.
 \end{aligned} \tag{2.13}$$

² I mean here an iterative procedure of choosing a new base point, direction, and optimization step. I shall illustrate it in detail in the next chapter (Chapter 3).

Instead of the discrete values $z_i^{(d)}$ their indexes i are used. Now, the discrete variable $Z^{(d)}$ can be handled as an integer one with boundary constraints ($1 \leq i \leq l$). For the evaluation of the objective function the discrete value itself is used in place of its index. Thus, discrete variable optimization is reduced to the integer variable one and discrete values are used only for objective function evaluation.

This approach showed the best results among evolutionary algorithms for mixed variable problems in mechanical engineering design [Lam99].

2.6 Constraints

Let the optimization problem be presented in the generally used form (2.14).

$$\begin{aligned}
 &\text{find} \quad X^* : f(X^*) = \min_X f(X) \\
 &\text{subject to} \\
 &\quad \text{boundary constraints} \quad L \leq X \leq H \\
 &\quad \text{constraint functions} \quad g_k(X) \leq 0, \quad k = 1, \dots, m
 \end{aligned} \tag{2.14}$$

2.6.1 Boundary Constraints

Boundary constraints represent low and high limits put on each individual:

$$L \leq \omega \leq H. \tag{2.15}$$

It is necessary that new values of variables satisfy the constraints after differentiation (or reproduction). For that, the values that have broken range conditions are randomly put back inside their limits.

$$\omega_j = \begin{cases} \text{rand}_j \cdot (h_j - l_j) + l_j & \text{if } \omega_j \notin [l_j, h_j] \\ \omega_j & \text{otherwise} \end{cases} \tag{2.16}$$

$j = 1, \dots, D.$

For integer variables one uses the next modification of this equation:

$$\omega_j = \begin{cases} \text{rand}_j \cdot (h_j - l_j + 1) + l_j & \text{if } \lfloor \omega_j \rfloor \notin [l_j, h_j] \\ \omega_j & \text{otherwise} \end{cases}. \tag{2.17}$$

In addition to the reinitialization (2.16) there are also other ways of boundary constraint handling. For example:

- Repeating of differentiation (2.8) until the trial individual satisfies the boundary constraints,

- Or, use of the periodic mode or the shifting mechanism proposed in [ZX03, MCTM04, PSL05],
- Or else, taking into consideration that boundary constraints are inequality constraints ($L - X \leq 0$ and $X - H \leq 0$), constraint handling methods and their modifications developed for constraint functions are also suitable for handling boundary constraints.

2.6.2 Constraint Functions

Penalty Function Method

I shall represent here a penalty function method for constraint handling used by Lampinen and Zelinka [LZ99b, LZ99c]. Compared with hard constraint handling methods, where infeasible solutions are rejected, the penalty function method uses penalties for moving into the feasible area M (2.2). Such penalties are directly added into the objective function. We give it in a logarithmic form:

$$\begin{aligned} \log \tilde{f}(\omega) &= \log(f(\omega) + a) + \sum_{k=1}^m b_k \cdot \log c_k(\omega) \\ c_k(\omega) &= \begin{cases} 1.0 + s_k \cdot g_k(\omega) & \text{if } g_k(\omega) > 0 \\ 1.0 & \text{otherwise} \end{cases} \\ s_k &\geq 1.0 \\ b_k &\geq 1.0 \\ \min f(\omega) + a &> 0. \end{aligned} \tag{2.18}$$

It is necessary that the objective function take only nonnegative values. For this reason the constant a is added. Even if the constant a takes too high values it does not affect the search process. The constant s scales constraint function values. The constant b modifies the shape of the optimizing surface. When the function value for a variable that lies outside a feasible area is insignificant, it is necessary to increase the values s and b . Usually, satisfactory results are achieved with $s = 1$ and $b = 1$.

It is clear that this method demands the introduction of extra control parameters, and therefore, in order to choose their effective values additional efforts are necessary. Generally, it is realized by trial and error, when the algorithm is started repeatedly for many times under various parameter values (s , b). It is obvious that this is not effective enough, so researchers are continuing investigations in this domain.

Modification of the Selection Operation

An original approach for constraint problem solution has been proposed in [Lam01, MCTM04]. The selection rule modification (2.10), where there is no need of using the penalty functions, has been shown there.

The basic idea is applying multiobjective optimization for handling constraints. This idea, it seems, was first communicated by David Goldberg as early as 1992 [Deb01] (pp.131–132). Later, three of its instances sequentially were reported to wider audience: (1) Coello Coello [Coe99b], (2) Deb [Deb00], and (3) Lampinen [Lam01]. Below, I shall describe Lampinen’s instance [Lam01], which is based on pure pareto-dominance defined in constraint function space.³

The choice of individual results from the next three rules.

- If both solutions ω and ind are feasible, preference is given to the lower objective function solution.
- The feasible solution is better than the infeasible.
- If the both solutions are infeasible, preference is given to the less infeasible solution.

Mathematically these rules are written as:

$$ind = \begin{cases} \omega & \text{if } \Phi \vee \Psi \\ ind & \text{otherwise} \end{cases},$$

where

$$\begin{aligned} \Phi &= [\forall k \in \{1, \dots, m\} : g_k(\omega) \leq 0 \wedge g_k(ind) \leq 0] \wedge \\ &\quad \wedge [f(\omega) \leq f(ind)] \\ \Psi &= [\exists k \in \{1, \dots, m\} : g_k(\omega) > 0] \wedge \\ &\quad \wedge [\forall k \in \{1, \dots, m\} : \max(g_k(\omega), 0) \leq \max(g_k(ind), 0)] . \end{aligned} \tag{2.19}$$

Thus, the trial vector ω will be chosen if:

- It satisfies all the constraints and provides lower objective function value or
- It provides lower than or equal to ind value for all constraint functions.

Notice that in the case of an infeasible solution, the objective function is not evaluated.

To prevent stagnation [LZ00], when the objective function values of both trial and target vectors are identical, preference is given to the trial one. In Appendix B you can find (proposed by me) the *C* source code of the above-described selection rules.

Other Constraint-Handling Methods

Finally I shall present a general classification of the constraint-handling methods for evolutionary algorithms. More detailed information can be found in [MS96, Coe99a, Coe02].

³ Discussed in personal communication with J. Lampinen.

1. Methods based on preserving feasibility of solutions
 - Use of specialized operators (Michalewicz and Janikow, 1991)
 - Searching the boundary of feasible region (Glover, 1977)
2. Methods based on penalty functions
 - Method of static penalties (Homaifar, Lai and Qi, 1994)
 - Method of dynamic penalties (Joines and Houck, 1994)
 - Method of annealing penalties (Michalewicz and Attia, 1994)
 - Method of adaptive penalties (Bean and Hadj-Alouane, 1992)
 - Death penalty method (Bäck, 1991)
 - Segregated genetic algorithm (Le Riche, 1995)
3. Methods based on a search for feasible solutions
 - Behavioral memory method (Schoenauer and Xanthakis, 1993)
 - Method of superiority of feasible points (Powell and Skolnick, 1993)
 - Repairing infeasible individuals (Michalewicz and Nazhiyath, 1995)

Problems

- 2.1.** What does evolutionary computation study?
- 2.2.** What three general trends of development of evolutionary algorithms do you know?
- 2.3.** Review once again problem (1.4). Indicate the “genes” of the individual.
- 2.4.** Usually, in differential evolution, the population is initialized by random values within boundary constraints. Propose your technique of initialization for the following cases: (a) there are no boundary constraints; (b) you have one or two solutions, but you do not know if these solutions are optimal; (c) your constraint handling method requires only feasible individuals and you need to preserve the uniformity of initialization. Implement and test the proposed techniques.
- 2.5.** Evaluate the fitness of the function $f(X) = e^{x_1 x_2 x_3 x_4 x_5} - \frac{1}{2}(x_1^3 + x_2^3 + 1)^2$ for the individual $X_0 = (-1.8, 1.7, 1.9, -0.8, -0.8)$.
- 2.6.** Which of the Evolutionary Algorithm’s operations is/are, most probably, time-consuming? (a) selection, (b) crossover, (c) mutation, (d) differentiation, (e) variation, (f) evaluation, (g) replacement, (h) recombination, (i) initialization. Explain your answer.
- 2.7.** What elements are common for all evolutionary algorithms?
- 2.8.** What is the difference between iteration, generation and evolutionary cycle?
- 2.9.** What is the stopping condition? Propose three different ways to end the optimization. Add it in your differential evolution and test.
- 2.10.** You are given the following maximization problem.

$$\begin{aligned} \max \quad & \frac{ax_1^4 x_3^2}{\pi^2 x_2^3 x_4} - \cos^2\left(2\pi d \frac{x_5}{x_3}\right) + e^{b \sin(2x_1)/x_2^3} - 3 \ln\left(c \frac{\pi}{4} x_2^2\right) + x_1 x_5 \\ & x_1 + x_2 + x_3 + x_4 \leq x_5 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \\ & a \geq 0, b < 0, c > 0, d \leq 0 \end{aligned}$$

Transform this problem into a minimization problem.

- 2.11.** How many optima has the function $|\sin(x)|$ in the range from 0 to 10?
- 2.12.** Give a definition of local optimum.
- 2.13.** Does the function given in problem (2.11) have a global optimum?

2.14. How does one calculate the trial individual? Show the concrete formula and give an explanatory sketch.

2.15. Determine and explain what is a “parent” and what is a “child” in differential evolution?

2.16. The crossover operation executes the inheritance of genes from the old to the new individual. Develop and implement your own crossover instead of the formula (2.9) proposed in Chapter 2.

2.17. What is the minimal size of the population you can use according to the formula you demonstrated in problem (2.14)?

2.18. Does differential evolution obey the natural selection theory of Darwin? What are the common and distinguishing features?

2.19. Is it possible, in differential evolution, that the “child” becomes “parent” in one and the same generation?

2.20. Find four distinctions between the classical DE (famous algorithm) and the neoteric one.

2.21. Recall and explain three principal advantages of neoteric DE.

2.22. Explain how DE handles integer variables? What is the advantage as against gradient methods?

2.23. Solve by hand the Traveling Salesman Problem with Time Windows. A truck driver must deliver to 9 customers on a given day, starting and finishing in the depot. Each customer $i = 1, \dots, 9$ has a time window $[b_i, e_i]$ and an unloading time u_i . The driver must start unloading at client i during the specified time interval. If he is early, he has to wait till time b_i before starting to unload. Node 0 denotes the depot, and c_{ij} the time to travel between nodes i and j for $i, j \in \{0, 1, \dots, 9\}$. The data are $u = (0, 1, 5, 9, 2, 7, 5, 1, 5, 3)$, $b = (0, 2, 9, 4, 12, 0, 23, 9, 15, 10)$, $e = (150, 45, 42, 40, 150, 48, 96, 100, 127, 66)$, and

$$(c_{ij}) = \begin{pmatrix} - & 5 & 4 & 4 & 4 & 6 & 3 & 2 & 1 & 8 \\ 7 & - & 2 & 5 & 3 & 5 & 4 & 4 & 4 & 9 \\ 3 & 4 & - & 1 & 1 & 12 & 4 & 3 & 11 & 6 \\ 2 & 2 & 3 & - & 2 & 23 & 2 & 9 & 11 & 4 \\ 6 & 4 & 7 & 2 & - & 9 & 8 & 3 & 2 & 1 \\ 1 & 4 & 6 & 7 & 3 & - & 8 & 5 & 7 & 4 \\ 12 & 32 & 5 & 12 & 18 & 5 & - & 7 & 9 & 6 \\ 9 & 11 & 4 & 12 & 32 & 5 & 12 & - & 5 & 22 \\ 6 & 4 & 7 & 3 & 5 & 8 & 6 & 9 & - & 5 \\ 4 & 6 & 4 & 7 & 3 & 5 & 8 & 6 & 9 & - \end{pmatrix}$$

2.24. Find an efficient model of problem (2.23) for solving by differential evolution. Focus attention on data representation, especially on realization of the permutation of clients and the fitness function. Solve the problem by DE and compare the results.

2.25. Solve problem (1.4) supposing that d and D take only integer values.

2.26. Write a code to handle discrete variables. Apply it to solving the problem (1.4) as is.

2.27. Formulate the problem of placing N queens on an N by N chessboard such that no two queens share any row, column, or diagonal. Use binary variables.

2.28. Could DE optimize binary variables? If yes, write the proper code and solve the N-queens problem (2.27) for $N = 4, 8, 16, \dots$. Otherwise, use a permutation to model the problem and solve it in integer variables. Think about an efficient method of constraints handling. Compare the results and determine which of two methods is more clever. Explain why.

2.29. What are the boundary constraints? What methods to handle boundary constraints do you know? Point out at least four methods and explain by giving an example.

2.30. Elaborate your own method of boundary constraints handling. Estimate its influence on the convergence of algorithms using the test functions from Appendix C.

2.31. The solution to a system of nonlinear equations specified by a mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a vector $X \in \mathbb{R}^n$ such that $f(X) = 0$. Algorithms for systems of nonlinear equations usually approach this problem by seeking a local minimizer to the problem

$$\min \{ \|f(X)\| : L \leq X \leq H \},$$

where $\|\cdot\|$ is some norm on $\mathbb{R}^n$, most often the l_2 norm. Solve any reasonable system of nonlinear equations using your own method of handling boundary constraints.

2.32. What is the penalty function? Create the penalty function for problem (1.4).

2.33. Solve problem (2.32). Experiment on the parameters s_k and b_k of the penalty function. Try to find their optimal values. Estimate its influence on the algorithm performance.

2.34. What drawbacks do you see in using penalty methods?

2.35. Try to implement independently a modification of the selection operation. Solve problem (1.4) with this method. Compare the results with ones obtained in problem (2.33).

2.36. Given D electrons, find the equilibrium state distribution of the electrons positioned on a conducting sphere. This problem, known as the Thomson problem of finding the lowest energy configuration of D point charges on a conducting sphere, originated with Thompson's plum pudding model of the atomic nucleus. The potential energy for D points (x_i, y_i, z_i) is defined by

$$f(x, y, z) = \sum_{i=1}^{D-1} \sum_{j=i+1}^D ((x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2)^{-\frac{1}{2}},$$

and the constraints on the D points are

$$x_i^2 + y_i^2 + z_i^2 = 1, \quad i = 1, \dots, D.$$

The number of local minima increases exponentially with D . Theoretical results show that

$$\min\{f(p_1, \dots, p_D) : \|p_i\| = 1, 1 \leq i \leq D\} \geq \frac{1}{2} D^2 (1 - \epsilon), \quad 0 \leq \epsilon \leq \left(\frac{1}{D}\right)^{1/2}.$$

Solve this problem for $D = 3, 10, 50$. How do you handle the nonlinear equality constraints? Are you far from the theoretical results?

2.37. Choose from the list of constraint-handling methods at the end of Subsection 2.6.2 any method you please and implement it.