Opposition-Based Differential Evolution

by

Shahryar Rahnamayan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2007

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.
I understand that my thesis may be made electronically available to the public.

Shahryar Rahnamayan

Abstract

Evolutionary algorithms (EAs) are well-established techniques to approach those problems which for the classical optimization methods are difficult to solve. Tackling problems with mixed-type of variables, many local optima, undifferentiable or non-analytical functions are some examples to highlight the outstanding capabilities of the evolutionary algorithms. Among the various kinds of evolutionary algorithms, differential evolution (DE) is well known for its effectiveness and robustness. Many comparative studies confirm that the DE outperforms many other optimizers. Finding more accurate solution(s), in a shorter period of time for complex black-box problems, is still the main goal of all evolutionary algorithms.

The opposition concept, on the other hand, has a very old history in philosophy, set theory, politics, sociology, and physics. But, there has not been any opposition-based contribution to optimization. In this thesis, firstly, the opposition-based optimization (OBO) is constituted. Secondly, its advantages are formally supported by establishing mathematical proofs. Thirdly, the opposition-based acceleration schemes, including opposition-based population initialization and generation jumping, are proposed. Fourthly, DE is selected as a parent algorithm to verify the acceleration effects of proposed schemes. Finally, a comprehensive set of well-known complex benchmark functions is employed to experimentally compare and analyze the algorithms. Results confirm that opposition-based DE (ODE) performs better than its parent (DE), in terms of both convergence speed and solution quality.

The main claim of this thesis is not defeating DE, its numerous versions, or other optimizers, but to introduce a new notion into nonlinear continuous optimization via innovative metaheuristics, namely the notion of opposition. Although, ODE has been compared with six other optimizers and outperforms them overall.

Furthermore, both presented experimental and mathematical results conform with each other and demonstrate that opposite points are more beneficial than pure random points

for black-box problems; this fundamental knowledge can serve to accelerate other machine learning approaches as well (such as reinforcement learning and neural networks). And perhaps in future, it could replace the pure randomness with random-opposition model when there is no a priori knowledge about the solution/problem.

Although, all conducted experiments utilize DE as a parent algorithm, the proposed schemes are defined at the population level and, hence, have an inherent potential to be utilized for acceleration of other DE extensions or even other population-based algorithms, such as genetic algorithms (GAs). Like many other newly introduced concepts, ODE and the proposed opposition-based schemes still require further studies to fully unravel their benefits, weaknesses, and limitations.

Acknowledgements

First of all, special thanks go to my cousin, Jila Roshanzamir, who is my best friend as well, for her endless and unbelievable pure kindness. Without her wide spread supports, this work would have never been possible.

No need to say, without my dear parents appreciation and encouragement, this work was not accomplishable. During this program, my father passed away. However, I could feel his attendance in my oral defense. I wish peace for his sprit and solace.

I would like to express the deepest thanks to my supervisors, Professor Hamid R. Tizhoosh and Professor Magdy M.A. Salama, for their continuous support and guidance. I have learned many academics and ethics matters from them.

Thanks to my thesis committee members for their constitutive comments and guidelines for the current and also future work.

I would like to thank all anonymous referees for their detailed and valuable comments on my published papers which helped me to improve the quality of this work. In particular, thanks to K. Price, the father of DE, for showing his impression about this work which encouraged me to go further and do my best.

I would like to thank Dr. Seyed Zia Al Din Sadr Al Ashrafi for offering beneficial knowledge about the footprint of the opposition in various sciences.

My thanks go to one-by-one of the colleagues and the staffs at the Systems Design Engineering Department, Vicky Lawrence in particular, for her kindly unlimited help to solve any sort of student problems.

I am thankful for the financial support from the Canadian Institute of Health Research (CIHR) Fellowship, the Ontario Graduate Scholarship (OGS), the University of Waterloo President's Graduate Scholarship, and the Faculty of Engineering Graduate Scholarship.

Finally, I would like to thank all my friends who did whatever they could to help me to accomplish this work.

To All Human Rights Activists

in general

and

Mother Tongue Education Activists

in particular

Contents

List of Tables

List of Figures

List of Acronyms

ACO ant colony optimization

AR acceleration rate

CEP classical evolutionary programming

CGA continuous GA

CI confidence interval

DE differential evolution

EP evolutionary programming

FADE fuzzy adaptive differential evolution

FEP fast evolutionary programming

FS free search

GAs genetic algorithms

NFC number of fitness function calls

OBL opposition-based learning

OBO opposition-based optimization

ODE opposition-based DE

PSO particle swarm optimization

QODE quasi-oppositional DE

SADE self-adaptive differential evolution

SP success performance

SR success rate

Std Dev standard deviation

VTR value to reach

Chapter 1

Introduction

God created suffering and heartache, so that joy might be known as their opposite. Hidden things become manifest through their opposites. But God has no opposite; so he remains hidden. Light is known as the opposite of darkness. But God's light has no opposite. Thus we cannot know him through our eyes.

– Rumi (1207 – 1273) in "Masnawi"

We are witness of the rapidly growing application of the population-based algorithms to solve real-world nonlinear complex problems. The computational time of these algorithms depend directly on the dimensionality and complexity of the problem. Although, compared to traditional methods, these algorithms are more applicable to real-world problems, they suffer from low convergence speed. So far, many attempts have been conducted to make them as fast as possible. Recently a new direction has been opened for this purpose, namely employing the opposition concept to make population-based algorithms faster.

1.1 Motivation

The footprints of the opposition concept can be observed in many areas around us. This concept has sometimes been labeled by different names. Opposite particles in physics, antonyms in languages, complement of an event in probability, antithetic variables in simulation, opposite proverbs in culture, absolute or relative complement in set theory, subject and object in philosophy of science, good and evil in animism, opposition parties in politics, theses and antitheses in dialectic, opposition day in parliaments, and dualism in religions and philosophies are just some examples to mention (Table 1.1 contains more examples and corresponding details). The Yin-Yang symbol in ancient Chinese philosophy is probably the oldest opposition concept which was expressed by human kind (Figure 1.1). Black and white represent yin (receptive, feminine, dark, passive force) and yang (creative, masculine, bright, active force), respectively. This symbol reflects the twisted duality of all things in nature, namely, receptive vs. creative, feminine vs. masculine, dark vs. bright, and finally passive vs. active forces. Even Greek classical elements to explain patterns in nature (Figure 1.2) mention the opposition concept, namely, fire (hot and dry) vs. water (cold and wet), earth (cold and dry) vs. air (hot and wet). Cold, hot, wet, dry present the pair-wised opposite characteristics of these four elements.

It seems that without using the opposition concept, the explanation of different entities

Table 1.1: Footprints of opposition in different fields.

Example	**Field**	**Description**
Opposite Particles/ Elements	Physics	Such as magnetic poles (N and S), opposite polarities (+ and −), electron-proton in an atom, action-reaction forces in Newton's third law, and so on.
Antonyms	Language	A word that means the opposite of another word (e.g., hot/cold, fast/slow, top/down, left/right, day/night, on/off).
Antithetic Variables	Simulation	Antithetic (negatively correlated) pair of random variables used for variance reduction.
Opposite Proverbs	Culture	Two proverbs with the opposite advice or meaning (e.g., The pen is mightier than the sword. Actions speak louder than words.); proverb or its opposite pair offers an applicable solution based on specific situation or condition.
Complements	Set theory	a) Relative complement ($B - A = \{x \in B \| x \notin A\}$), b) Absolute complement ($A^c = U - A$, where U is the universal set).
Opposition Party	Politics	A political party or organized group opposed to the government.
Inverter	Digital design	Output of the inverter gate is one if input is zero and vice versa.
Opposition Day	Legislation	A day in the parliament in which small opposition parties are allowed to propose the subject for debate (e.g., Canada's parliament has 25 opposition days).
Dualism	Philosophy and Religion	Two fundamental principles/concepts, often in opposition to each other; such as "Yin" and "Yang" in Chinese philosophy and Taoist religion (Figure 1.1), "subject" and "object" in philosophy of science, "good" and "evil" in animism, "ahura" and "ahriman" in Zarathustra.
Dialectic	Philosophy	An exchange of "theses" and "antitheses" resulting in a "synthesis" (e.g. in Hinduism, these three elements are creation (Brahma), maintenance of order (Vishnu) and destruction or disorder (Shiva)).
Classical Elements	Archetype	A set of archetypal classical elements to explain patterns in nature (e.g., the Greek classical elements are Fire (hot and dry), Earth (cold and dry), Air (hot and wet), and Water (cold and wet), Figure 1.2).
if-then-else	Algorithm	if *condition* then *statements* [else *elsestatements*], the *else statements* are executed when the opposite of the *condition* happens.
Complement of an Event	Probability	$P(A') = 1 - P(A)$.
Revolution	Socio-political	A significant Socio-political change in a short period of time.

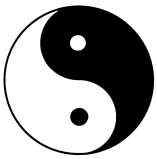


Fig. 1.1: Opposition concept embedded in the Yin-Yang symbol or Taijitu in ancient Chinese philosophy.

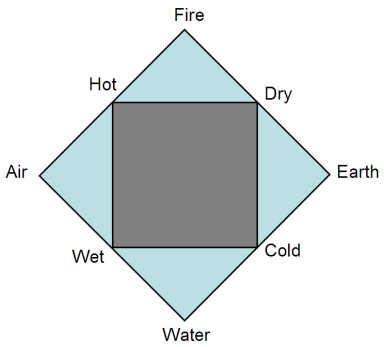


Fig. 1.2: Greek classical elements to explain patterns in nature are Fire (hot and dry), Earth (cold and dry), Air (hot and wet), and Water (cold and wet).

around us is hard and maybe even impossible. In order to explain an entity or a situation we sometimes explain its opposite instead. In fact, opposition often manifests itself in a balance between completely different entities. For instance, the east, west, south, and north can not be defined alone. The same is valid for cold and hot and many other examples. Extreme opposites constitute our upper and lower boundaries. Imagination of the infinity is vague, but when we consider the limited, it then becomes more imaginable because its opposite is definable.

Sometimes, we apply the opposition concept in our regular life unconsciously. Let us look at a simple example (see Figure 1.3). Suppose police officers want to arrest somebody in a Theater Hall. There are two seat groups (A and B) and entrance doors ($a-k$), on one side (Figure 1.3(a)). The seat position of the target person is unknown (like the position

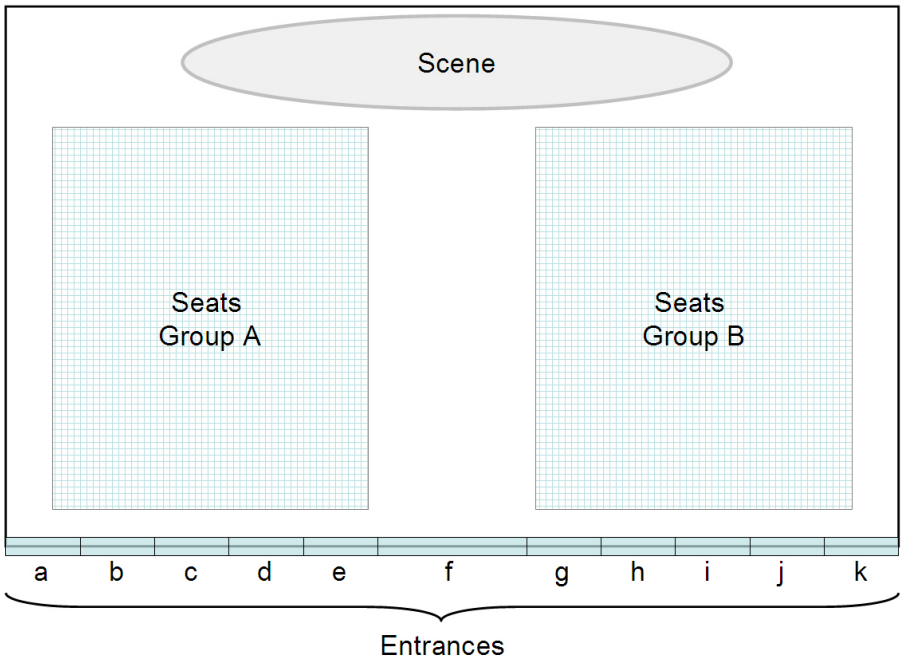
of the solution in a black-box optimization problem) and only two officers are available. If the first officer selects the door a, which door should be selected by the second one? What happens if the first officer selects the door b? If just one officer is available which door should be selected by him/her? For the last case since there is no opposite for one officer, entering from the middle door (f) supports the highest accessibility.

Let us consider the same example but this time a Theater Hall with entrance doors on all four sides (Figure 1.3(b)). Now, let us repeat the same questions for this situation. Increase the number of officers (like individuals in a population-based optimization method) and repeat the same questions. When officer one selects the door h, the second officer usually selects d. Why are the other doors not selected instead? It seems that even when we increase the number of officers the opposition pattern for covering the doors is still followed (e.g., north-west door is selected versus south-east door). These are officers' intuitive decisions in different situations, and perhaps they are unaware of the concept of the opposition but they apply it in order to cover the search space more efficiently.

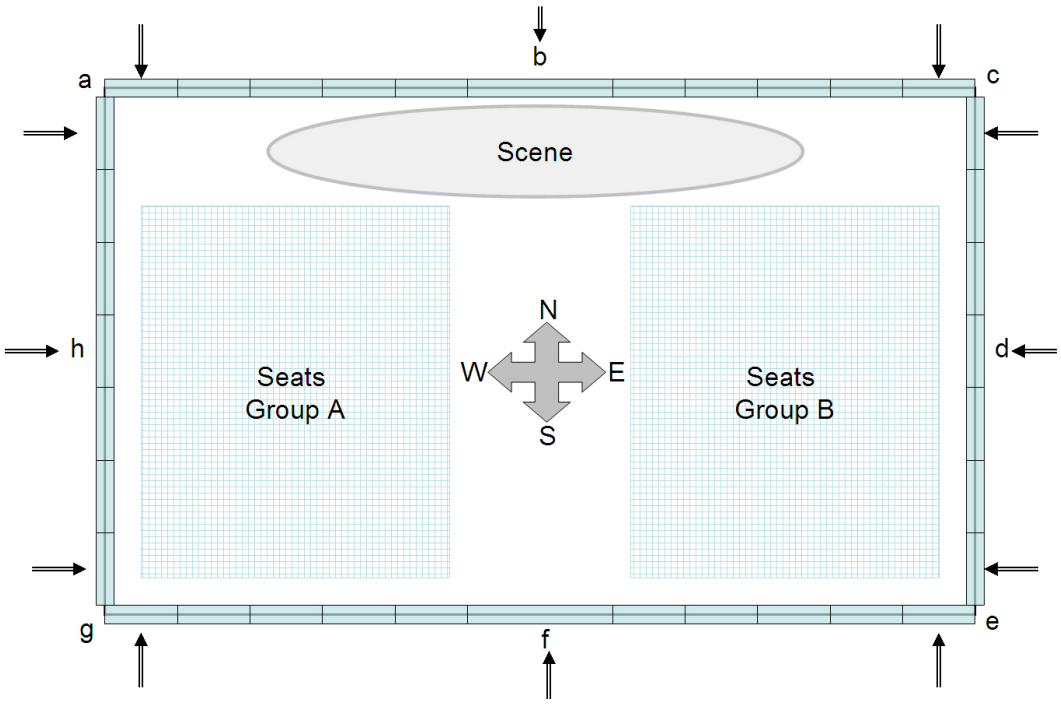
For the first time, the concept of opposition-based learning (OBL), in its earlier simple form, was introduced by Tizhoosh [2005b]. Then, it was applied to accelerate reinforcement learning [Tizhoosh 2005c, 2006]. The main idea behind OBL is the simultaneous consideration of an estimate and its corresponding opposite estimate (i.e., guess and opposite guess) in order to achieve a better approximation for the current candidate solution. As an advantage of opposite versus random points, purely random resampling or selection of solutions from a given population, has a higher chance of visiting or even revisiting unproductive regions of the search space.

Finding the more accurate solution(s) in a shorter period of time for complex nonlinear problems, is the main goal of all optimization metaheuristics and still widely open to research. All of these facts encourage us to employ the opposition concept to accelerate optimization techniques.

In this study, the population-based algorithms, differential evolution in particular, is



(a) Theater Hall with entrance doors on one side.



(b) Theater Hall with entrance doors on all four sides.

Fig. 1.3: Theater Hall example.

selected to be accelerated using the opposition concept. The main reason for selecting differential evolution is its high ability to find precise solutions for the mixed-typed black-box global optimization problems [Feoktistov 2006]. The population-based algorithms are computationally expensive and hence their acceleration is widely appreciated. Among the various evolutionary algorithms [Bäck 1996; Bäck et al. 1997; Schwefel 2003], Differential Evolution (DE) is well known for its effectiveness and robustness. Frequently reported studies demonstrate that the DE outperforms many other optimizers over both benchmark functions and real-world applications.

1.2 Scope and Objectives

The research in this thesis concentrates on the acceleration of the population-based algorithms in general, and the differential evolution algorithm in particular. According to the proposed classification scheme for optimization methods by Feoktistov [2006] (Figure 1.4), DE is a population-based nonlinear continuous global optimization algorithm. The optimization branch in focus of this research is highlighted by bolded arrows. The classical DE has been extended to tackle the mixed-type variable optimization problems [Lampinen and Zelinka 1999b]. The population-based branch covers a wide range of families, such as genetic algorithms (GAs) [Goldberg 1989], particle swarm optimization (PSO) [Angeline 1998], ant colony optimization (ACO) [Dorigo and Stützle 2004], differential evolution [Storn and Price 1997b], and free search (FS) [Penev and Littlefair 2005]. The focus of this research is on bound-constraints differential evolution algorithms.

The main objectives of this research are: 1) constituting the framework of the opposition-based optimization and corresponding definitions, theorems, and mathematical proofs which demonstrate why integration of the opposition concept is valuable in optimization, 2) proposing the opposition-based acceleration schemes for the population-based algorithms, 3) accelerating differential evolution by proposed opposition-based schemes, 4) benchmark-

ing and parameter analysis of the ODE by employing a comprehensive test suite of global optimization problems, 5) proposing possible enhancement directions for ODE. More details about the related contributions are provided in the final chapter.

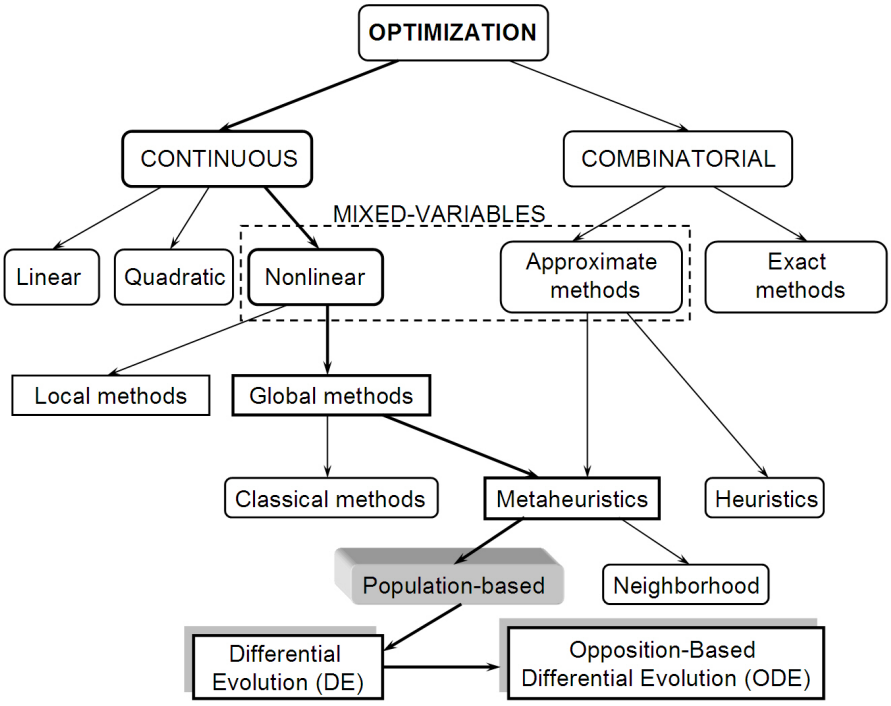


Fig. 1.4: A simple classification scheme of optimization methods [Feoktistov 2006]. According to this classification, DE is a population-based nonlinear continuous global optimization method. Opposition-based DE (ODE) is the enhanced version of the DE by the opposition concept. The optimization branch in focus of this research is highlighted by bolded arrows and boxes.

1.3 Outline of the Thesis

Chapter 2 explains the differential evolution algorithm and the main reasons to select it as a parent evolutionary algorithm to implement the proposed opposition-based schemes. Also,

this chapter reviews some studies, which compare DE to other well-known optimizers. It can be skipped if the reader is familiar with DE and its frequently reported higher performance on benchmark functions and real-world applications.

Chapter 3 covers the details of the opposition-based optimization and corresponding definitions, theorems, and proofs. This chapter presents answers to questions concerning why and how far the opposite points are more valuable than additionally generated random points. In order to support the mathematical computations, they are compared to the experimental results as well.

Chapter 4 presents the general schemes of the opposition-based algorithm. Then, these schemes are embedded inside differential evolution to introduce ODE. The overall features of the aforementioned schemes are explained in details.

Chapter 5 contains the empirical study and analysis of the proposed opposition-based differential evolution. A comprehensive set of experimental series, including parameter and comparative studies, are conducted in this chapter. ODE is compared with six other evolutionary optimizers.

Chapter 6 includes a new optimization-based image thresholding algorithm. It also compares micro-DE and micro-ODE on thresholding of test images. Finally, the conclusions, contributions, and future works all are addressed in Chapter 7.

Chapter 2

Differential Evolution (DE): A Short Review

Differential Evolution, deriving from population-based metaheuristics, inherits all the best properties of its ancestors: global methods of nonlinear continuous optimization, approximate methods of combinatorial optimization, mixed-variables handling, and so on. In addition, it provides stochastic optimization principles, distributed searchers, and universal heuristics. All this makes the algorithm a general-purpose optimizer, remaining, with all this going on, simple, reliable, and fast.

– Vitaliy Feoktistov, 2006 in "Differential Evolution: In Search of Solutions"

2.1 Introduction

Differential evolution (DE) is a population-based optimization algorithm. The invention of the DE algorithm goes back to Genetic Annealing by Kenneth Price [1994] and solving the Chebyshev polynomial fitting problem by him and Rainer Storn. In order to solve Chebyshev problem in continuous space, they modified the genetic annealing algorithm from bit-string to floating-point encoding and consequently switched from logical operators to arithmetic ones. During experiments, they discovered the differential mutation to perturb the population of vectors. They also noticed that by using differential mutation, discrete recombination, and pair-wise selection, there is no need to apply annealing mechanism; it was removed completely and DE was born. Results of this achievement were reported in the ICIS technical report in 1995 [Storn and Price 1995]. DE was published in the Dobb's Journal [Price and Storn 1997] and then in the Journal of Global Optimization [Storn and Price 1997b]. By this way, DE's capacity and advantages were introduced to the optimization community. Comprehensive history and development of DE is presented in literature [Feoktistov 2006].

2.2 Why Differential Evolution?

It is important to note why DE was selected as a parent algorithm in this thesis. Simplicity of the DE is the first reason on this track. The compactness of the algorithm and easy-to-understand steps, both impress any researcher or practitioner to work with the DE.

Working directly with the continuous variables (arithmetic operators instead of logical operators) is the second attraction feature for this evolutionary algorithm. Unlike many binary versions of the genetic algorithm, DE works with the floating-point numbers; encoding and decoding of the variables (which can be the source of inaccuracy) are removed. Consequently, this feature makes DE scalable for high dimensional problems and also time and

memory efficient. DE's steps are described using simple numerical and pictorial examples in Section 2.3.

DE does not need a probability density function (PDF) to adapt the control parameters (unlike most Evolutionary Strategies [Schwefel 1995; Fathi and Hildebrand 1997]) or any probability distribution pattern for the mutation (dissimilar to the genetic algorithm or evolutionary programming). DE's different mutation and crossover schemes separate it from other evolutionary algorithms. Section 2.7 explains different DE variants.

The ability of the handling mixed integer, discrete, and continuous variables using some straightforward approaches makes DE more realistic for a wide range of real-world applications [Lampinen and Zelinka 1999a,b]. Discrete variables can be assumed as a subset of integer variables. The main advantage of DE, during working with the integer variables, is that it internally works on continuous space and only switches to the integer space during the evaluation of the objective function. This characteristic supports higher accuracy compared to some other well-known algorithms (e.g., GAs) which perform in the reverse manner. More details about handling mixed-type variables are provided in Section 2.5.

Extensions of classical DE are capable of handling boundary constraints and also nonlinear function constraints which both are commonly required in the real-world problems. This case is covered in Section 2.6.

Most importantly, many comparative studies report the higher robustness, convergence speed, and solution quality of the DE. This is true over benchmark functions and also real-world applications. A comprehensive performance study is provided in Benchmarking Differential Evolution chapter in [Price et al. 2005]. In this chapter, they first compare DE to 16 other optimizers against five well-known thirty-dimensional test functions (namely, Rosenbrock, Ackley, Griewangk, Rastrigin, and Schwefel). For this study, DE's 16 competitors are:

- Evolutionary programming with adaptive Lévy mutations (ALEP) [Lee and Yao 2004]

- Attractive and repulsive particle swarm optimization (arPSO) [Vesterstroem and Thomsen 2004]
- Cooperative co-evolutionary genetic algorithm (CCGA) [Bergh and Englebrecht 2004]
- Classical evolutionary programming (CEP) [Yao et al. 1999]
- Conventional evolutionary programming with adaptive mutations (CEP/AM) [Chellapilla 1998]
- Classical evolution strategies (CES) [Yao and Liu 1997]
- Cooperative particle swarm optimization (CPSO-S6) [Bergh and Englebrecht 2004]
- Evolutionary optimization (EO) [Angeline 1998]
- Fast evolution programming (FEP) [Yao et al. 1999]
- Fast evolutionary strategies (FES) [Yao and Liu 1997]
- Hybrid taguchi-genetic algorithm (HTGA) [Tsai et al. 2004]
- Orthogonal genetic algorithm (OGA) [Leung and Wang 2001]
- Particle swarm optimization (PSO) [Angeline 1998]
- Quantum evolutionary algorithm with rotation (QEA/R) [Han and Kim 2004]
- Simple evolutionary algorithm (SEA) [Vesterstroem and Thomsen 2004]
- Stochastic genetic algorithm (StGA) [Tu and Lu 2004].

Then, they explored previously conducted eight benchmark-function-based comparative studies (namely, unconstraint optimization [Storn and Price 1997b; Paterlini and Krink 2004; Vesterstroem and Thomsen 2004; Ali and Törn 2004], multi-constraints nonlinear optimization [Lampinen 2004], mixed-variable optimization [Lampinen and Zelinka 1999b], multi-objective optimization [Kukkonen and Lampinen 2004], noisy-function optimization [Krink et al. 2004]) and also eleven application-oriented performance comparison studies (namely, multi-sensor fusion [Joshi and Sanderson 1999], earthquake relocation [Bohuslav and Michal 2001], DC operating point analysis for nonlinear circuits [Crutchley and Zwolinski 2004], estimation of heat transfer parameters in a trickle-bed reactor [Babu and Sastry 1999], aerodynamic optimization [Rogalsky et al. 1999], image registration [Salomon 2001], identifying induction motor parameters [Ursem and Vadstrup 2004], optimization of neural networks [Fischer et al. 1999; Plagianakos et al. 2001], and optimization of carbon and silicon cluster geometry [Ali and Törn 2000; Chakraborti et al. 2002]). Finally, they concluded as follows:

> "[...] DE may not always be the fastest method, it is usually the one that produces the best results, although the number of cases in which it is also the faster is significant. DE also proves itself to be robust, both in how control parameters are chosen and in the regularity with which it finds the true optimum. [...] As these researchers have found, DE is a good first choice when approaching a new and difficult global optimization problem is defined with continuous and/or discrete parameters. "

Although, DE is almost in its infancy (approx. 10 years old) and is being improved step-by-step, because of above mentioned reasons, it has established itself as a universal optimization tool. The sharp growing of the publications on DE confirms this fact clearly (see Figure 2.1).

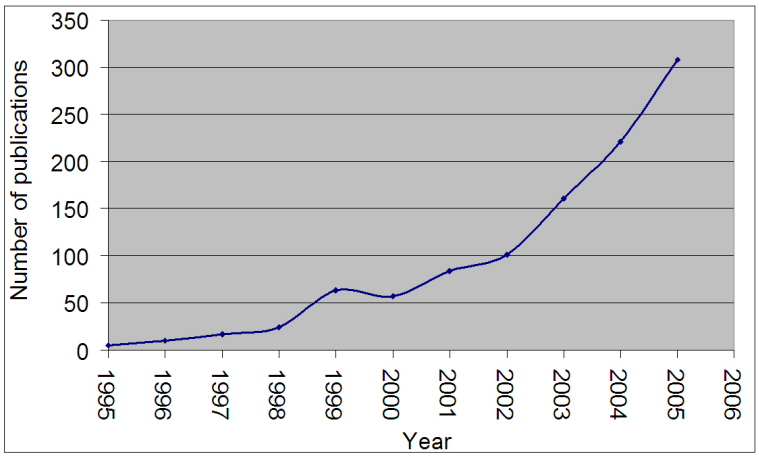


Fig. 2.1: Publications on Differential Evolution (based on information extracted from Google's Scholar service).

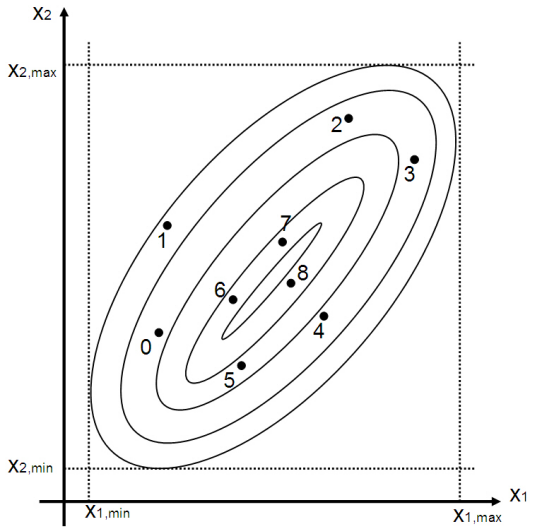
2.3 DE Algorithm

Differential Evolution is a population-based directed search method [Price 1999]. Like other evolutionary algorithms, it starts with an initial population vector, which is randomly generated when no preliminary knowledge about the solution space is available. Each vector of the initial population can be generated as follows [Price et al. 2005]:

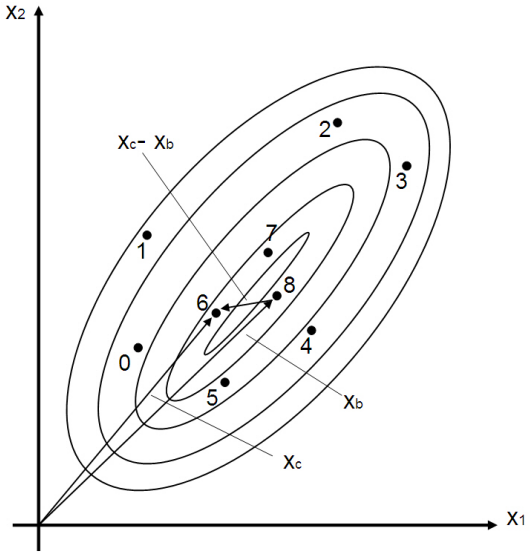
$$X_{i,j} = a_j + rand_j(0,1) \times (a_j - b_j); j = 1, 2, ..., D, \tag{2.1}$$

where D is the problem dimension; a_j and b_j are the lower and the upper boundaries of the variable j, respectively. $rand(0,1)$ is the uniformly generated random number in $[0,1]$. A sample initial population for ODE is shown in Figure 2.2(a).

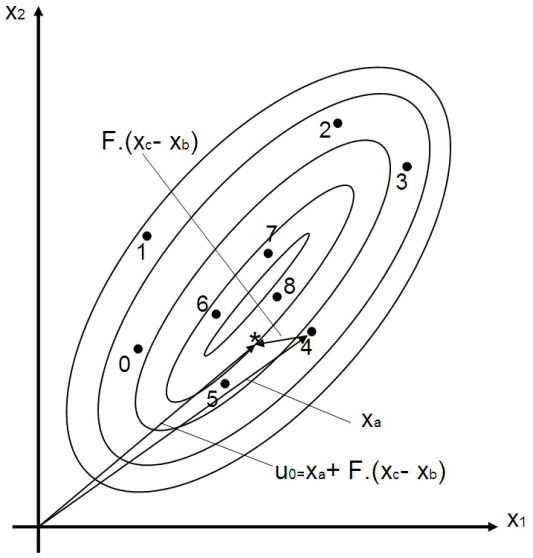
Let us assume that $X_{i,G}(i = 1, 2, ..., N_p)$ are candidate solution vectors in generation G (N_p : population size). Successive populations are generated by adding the weighted difference of two randomly selected vectors to a third randomly selected vector. For classical DE



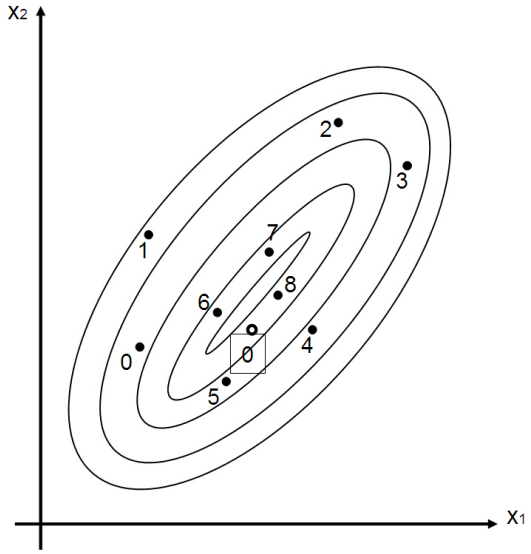
(a) Population initialization for DE ($N_p = 9$). Contour lines for $f(x_1, x_2)$ are shown by ellipses.



(b) Generating difference vector $X_c - X_b$. b and c are the randomly selected indices.



(c) Generating $X_{a,G} + F(X_{c,G} - X_{b,G})$. a is the third randomly selected index.



(d) After the crossover if the generated vector has lower objective value; it will be replaced with the vector 0.

Fig. 2.2: Illustration of one generate-and-test cycle for DE (starting from vector 0).

$(DE/rand/1/bin)$[1], the mutation, crossover, and selection operators are straightforwardly defined as follows:

Mutation - For each vector $X_{i,G}$ in generation G a mutant vector $V_{i,G}$ is defined by

$$V_{i,G} = X_{a,G} + F(X_{c,G} - X_{b,G}), \tag{2.2}$$

where $i = \{1, 2, ..., N_p\}$ and a, b, and c are mutually different random integer indices selected from $\{1, 2, ..., N_p\}$. Further, i, a, b, and c are different so that $N_p \geq 4$ is required. $F \in [0, 2]$ is a real constant which determines the amplification of the added differential variation of $(X_{c,G} - X_{b,G})$. Larger values for F result in higher diversity in the generated population and lower values cause faster convergence. Figures 2.2(b) and 2.2(c) illustrate vector representation of the $(X_{c,G} - X_{b,G})$ and $X_{a,G} + F(X_{c,G} - X_{b,G})$, respectively.

Crossover - DE utilizes the crossover operation to generate new solutions by shuffling competing vectors and also to increase the diversity of the population. For the classical DE $(DE/rand/1/bin)$, the binary crossover (shown by 'bin' in the notation) is utilized. It defines the following trial vector:

$$U_{i,G} = (U_{1i,G}, U_{2i,G}, ..., U_{Di,G}), \tag{2.3}$$

$$U_{ji,G} = \begin{cases} V_{ji,G} & \text{if } rand_j(0,1) \leq C_r \vee j = k, \\ X_{ji,G} & \text{otherwise.} \end{cases} \tag{2.4}$$

$C_r \in (0, 1)$ is the predefined crossover rate, and $rand_j(0, 1)$ is the j^{th} evaluation of a uniform random number generator. $k \in \{1, 2, ..., D\}$ is a random parameter index, chosen once for each i to make sure that at least one parameter is always selected from the

[1]This notation is explained in section 2.7.

mutated vector, $V_{ji,G}$. Most popular values for C_r are in the range of $(0.4, 1)$ [Das et al. 2005a]. Figure 2.3 presents a pictorial example for the binary crossover.

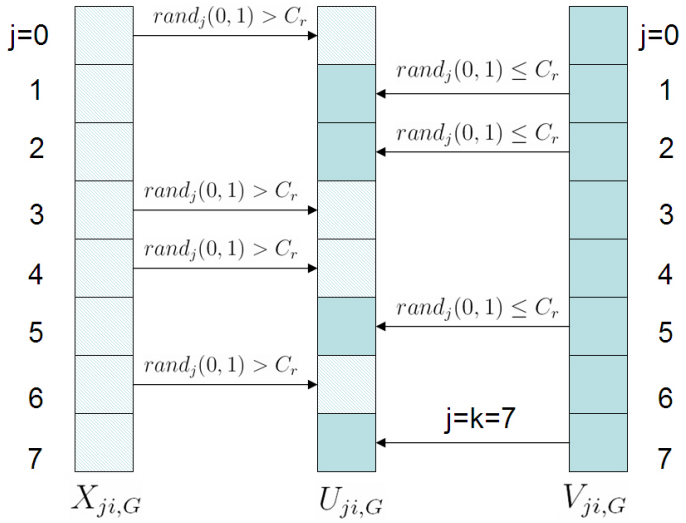


Fig. 2.3: A pictorial example for the binary crossover in DE ($k = 7$).

Selection - This is an approach which must decide which vector ($U_{i,G}$ or $X_{i,G}$) should be a member of next (new) generation, $G + 1$. For a minimization problem, the vector with the lower value of objective function is chosen (greedy selection).

This evolutionary cycle (i.e., mutation, crossover, and selection) is repeated N_p (population size) times to generate a new population. These successive generations are produced until meeting the predefined termination criteria. Algorithm 1 presents the details of the classical DE algorithm. Starting point for the mutation, crossover and selection is indicated by the comments in the algorithm. The algorithm terminates when the best achieved fitness value (BFV) is smaller than the value to reach (VTR), or the number of fitness function calls (NFC) exceeds predefined maximum number of function calls (MAX_{NFC}).

Termination strategy can be defined differently based on the problem nature, application, or the purpose of the experiment. Limiting the number of generations, the execution time, or checking some statistics of the population (e.g., diversity or the improvement rate) are some commonly used termination criteria.

2.4 A Numerical Example for DE

In order to illustrate the steps of the DE algorithm a typical numerical example is presented here [Onwubolu and Babu 2004]. Figure 2.4 shows the procedure of generating one vector for the next generation. The objective function is $f(X) = x_1 + x_2 + x_3 + x_4 + x_5$ and the dimension size, population size, mutation constant, and crossover constant are $5, 6, 0.8$ and 0.5, respectively. The procedure is similar to generate the rest of the next population's vectors. As seen, the current generation G has six individuals, $N_p = 6$; and each individual contains five parameters, $D = 5$ $(X(x_1, x_2, x_3, x_4, x_5))$. Cost value for each individual is calculated by $f(X)$ and is shown in the top cell of the corresponding vector. For the first target vector (individual 1), three mutually different vectors are selected randomly (individuals 2,4, and 6). Then, the noisy vector is calculated as follow:

$$\textit{noisy vector} = \textit{individual 6+ } F\times\textit{(individual 2 } - \textit{ individual 4)}$$

Consequently, with the crossover of the noisy vector and target vector the trial vector is generated. Parameters 1 and 5 are selected from noisy vector and the rest from the target vector. Now, in selection step, the cost value of the trial vector and the target vector is compared and the vector with the lower cost value (target vector here) is selected and copied to the next generation G+1.

Algorithm 1 Differential Evolution (DE). P_0: Initial population, N_p: Population size, V: Noise vector, U: Trial vector, D: Problem dimension, BFV: Best fitness value so far, VTR: Value-to-reach, NFC: Number of function calls, MAX_{NFC}: Maximum number of function calls, F: Mutation constant, $rand(0,1)$: Uniformly generated random number, C_r: Crossover rate, $f(\cdot)$: Objective function, P': Population of the next generation.

Generate uniformly distributed random population P_0
while (BFV $>$ VTR **and** NFC $<$ MAX_{NFC}) **do**
  //Generate-and-Test-Loop
  for $i = 0$ to N_p **do**
    Select three parents X_a, X_b, and X_c randomly from current population where $i \neq a \neq b \neq c$
    //Mutation
    $V_i \leftarrow X_a + F \times (X_c - X_b)$
    //Crossover
    for $j = 0$ to D **do**
      if $rand(0,1) < C_r$ **then**
        $U_{i,j} \leftarrow V_{i,j}$
      else
        $U_{i,j} \leftarrow X_{i,j}$
      end if
    end for
    //Selection
    Evaluate U_i
    if $(f(U_i) \leq f(X_i))$ **then**
      $X'_i \leftarrow U_i$
    else
      $X'_i \leftarrow X_i$
    end if
  end for
  $X \leftarrow X'$
end while

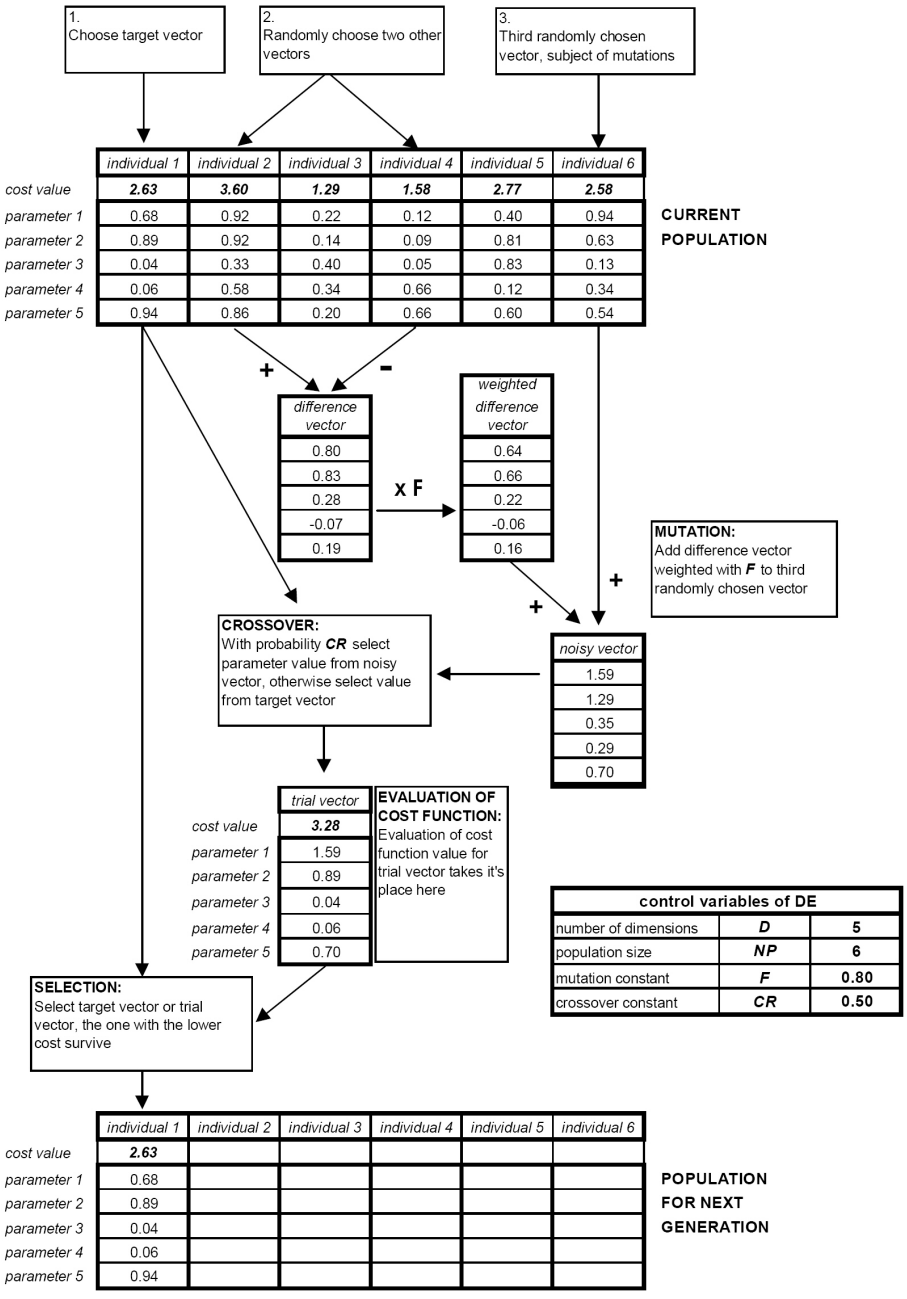


Fig. 2.4: A numerical example [Onwubolu and Babu 2004] to illustrate the classical DE ($DE/rand/1/bin$) for minimizing a simple objective function, $f(X) = x_1+x_2+x_3+x_4+x_5$.

2.5 Handling Integer and Discrete Variables

DE works directly with floating-point variables, but by applying some minor modifications it is able to handle integer variables as well. One of the common ways is simply using the integer variables during evaluation of the objective function and leaving other parts unchanged [Onwubolu and Babu 2004]. Following rules present this approach:

$$f(y_j);\ j = 1, ..., D$$

$$y_j = \begin{cases} x_j & x_j \text{ is a continuous variable,} \\ \text{INT}(x_j) & x_j \text{ is an integer variable.} \end{cases} \tag{2.5}$$

or

$$y_j = \begin{cases} x_j & x_j \text{ is a continuous variable,} \\ \text{ROUND}(x_j) & x_j \text{ is an integer variable.} \end{cases} \tag{2.6}$$

Converting a continuous number to the integer can be performed by INT($\cdot$) or ROUND($\cdot$) functions. INT($\cdot$) truncates the real part of the continuous number and ROUND($\cdot$) returns the closest integer number. If the INT($\cdot$) is the conversion case, Eq. 2.1 should be replaced with the following equation because of truncation effect of INT($\cdot$) function:

$$X_{i,j} = a_j + rand_j(0,1) \times (a_j - b_j + 1), j = 1, 2, ..., D. \tag{2.7}$$

Through this method, DE internally searches a superset of the solution space instead, because the integer numbers are the subset of the continuous numbers. This feature increases the population diversity and the robustness of the algorithm as well.

In order to handle discrete variables, the method performs similar to the integer case, but this time the indices of the discrete numbers are used as integer variables and during

the objective function evaluation, instead of the integer value of the indices, the discrete value of the variables is substituted.

2.6 Boundary Constraints Handling

DE's mutation operator can expand the search space to the outside of the predefined boundaries. For unconstraint boundary problems it can be an advantage, because even solutions, which are outside the boundaries, can be explored. But, in the revers sense, for the boundary constraint problems this phenomenon should be prevented. An example of the violation by the mutation operation is presented in Figure 2.5. Two approaches are commonly applied to prevent this violation [Onwubolu and Babu 2004] (a) generating a new random value (re-initialization) for the parameters which violate the boundary constraints or (b) re-performing the mutation operation to achieve a vector with no violating parameter. The first approach can be faster than the second one in overall, because the second method sometimes needs more than one time repeating the mutation to obtain an acceptable vector.

2.7 DE's Variants and Notations

Classical DE ($DE/rand/1/bin$) was explained in the previous sections. There are other DE variants which are indicated by the notation $DE/a/b/c$ [Mezura-Montes et al. 2006]. In this notation, 'a' specifies the vector which should be mutated; it can be the best vector ('*best*') of the current population or a randomly selected one ('*rand*'). 'b' is reserved for the number of difference vectors which participate in the mutation (1 or 2) and 'c' denotes the applied crossover scheme, binary ('*bin*') or exponential ('*exp*'). Figure 2.6 presents a pictorial example for the exponential crossover. Crossover procedure starts from a randomly selected parameter ($j = 3$ in this example), and copies the parameters of the

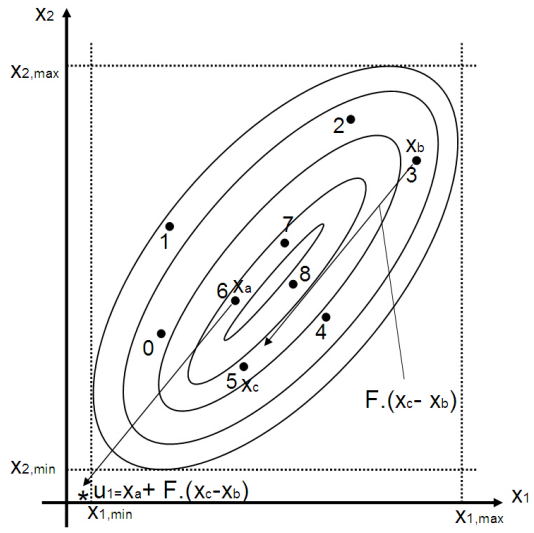


Fig. 2.5: An example of the violation by the mutation operation from the predefined boundary constraints. u_1 (marked by '$*$') is located outside the box shown by $[x_{1,min}, x_{1,max}], [x_{2,min}, x_{2,max}]$.

$U_{ji,G}$ to $V_{ji,G}$, until the first occurance of $rand(0,1) > C_r$; then the remaining parameters from $X_{ji,G}$ are inherited by $V_{ji,G}$.

As an example, $DE/best/2/exp$ specifies a DE scheme with exponential crossover and the following mutation:

$$V_{i,G} = X_{best,G} + F(X_{a,G} - X_{b,G}) + F(X_{c,G} - X_{d,G}), \tag{2.8}$$

where $X_{best,G}$ is the best vector in the population. $X_{a,G}$, $X_{b,G}$, $X_{c,G}$, and $X_{d,G}$ are four different randomly selected vectors ($a \neq b \neq c \neq d$) from the current population.

Other studies have been conducted to enhance the performance of the classical DE algorithm, e.g, adaptively determining DE control parameters. The fuzzy adaptive differential evolution algorithm (FADE) was introduced by Liu and Lampinen [2005]. They

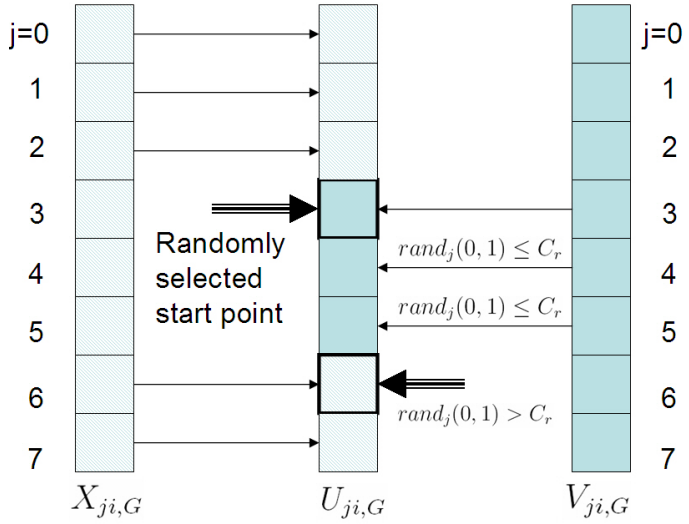


Fig. 2.6: A pictorial example for the exponential crossover in DE.

employed a fuzzy logic controller to set the mutation and crossover rates. In the same direction, Brest et al. [2006] proposed self-adaptive differential evolution (SADE). Teo [2006] proposed a dynamic population sizing strategy based on self-adaptation and Ali and Törn [2004] introduced auxiliary population and automatic calculating of the amplification factor, F, for the difference vector.

Other researchers have experimented with multi-population ideas. Tasoulis et al. [2004] proposed parallel DE where they assign each subpopulation to a different processor node. Shi et al. [2005] partitioned high-dimensional search spaces into smaller spaces and used multiple cooperating subpopulations to find the solution. They called this method cooperative co-evolutionary differential evolution.

Hybridization with different algorithms is another direction for improvement. Sun et al. [2005] proposed a new hybrid algorithm based on a combination of DE and estimation of distribution algorithm. This technique uses a probability model to determine promising regions in order to focus the search process on those areas. Noman and Iba [2005] incor-

porated local search into the classical DE. They employed fittest individual refinement which is a crossover-based local search. Fan and Lampinen [2003] introduced a new local search operation, trigonometric mutation, in order to obtain a better trade-off between convergence speed and robustness. Kaelo and Ali [2006] employed reinforcement learning and different schemes for generating fitter trial points. For more details about DE extensions, variants, and species the reader is referred to literature [Onwubolu and Babu 2004; Price et al. 2005; Feoktistov 2006].

2.8 Summary

In this chapter, the main reasons to select DE as a parent algorithm as well as its classical version were briefly discussed. Different DE variants with respect to its mutation strategy and crossover procedure, were explained and illustrated with some numerical and pictorial examples. Methods to handle integer/discrete variables and also applying the boundary constraints, were reviewed shortly.

According to the reported results from many comparative studies, DE presents itself as a strong candidate to be a universal optimizer. Rapidly growing publications strengthen this hope. DE works directly with the continuous space but its extended versions are capable to tackle mixed-type variable problems. Its capability to handle the constraint functions as well as multiobjective problems attracts more researchers and practitioners.

DE's simplicity is another important advantage which helps us to implemented it by very compact algorithms. The presented DE algorithm in this chapter (Algorithm 1) is utilized as a parent algorithm for introducing any other enhanced version.

Chapter 3

Opposition-Based Optimization

What is wanted is not the will to believe, but the wish to find out, which is the exact opposite.

– Bertrand Russell

Mathematics is a study which, when we start from its most familiar portions, may be pursued in either of two opposite directions.

– Bertrand Russell in "Introduction to Mathematical Philosophy"

3.1 Introduction

In this chapter definitions, theorems, and proofs related to opposite points and the opposition - based optimization are presented. The benefit of the opposite points (compared to independent random inputs) in one dimensional space is investigated through a mathematical proof. Then, this proof is extended to D-dimensional space. Furthermore, the probability of the opposite points being closer to the solution is calculated mathematically and verified experimentally; both results support each other. The main concern of the presented proofs is black-box optimization problems. Finally, this chapter wants to answer the following question, in general form: Why are opposite points valuable?

3.2 Opposite Points in Continuous Space

3.2.1 Definitions

Definition 3.1 *Let x be a real number in an interval $[a, b]$ ($x \in [a, b]$); the opposite of x, denoted by $\breve{x}$, is defined by*

$$\breve{x} = a + b - x. \tag{3.1}$$

Figure 3.1 (top) illustrates x and its opposite $\breve{x}$ in interval $[a, b]$. As seen, x and $\breve{x}$ are located in equal distance from the interval's center ($|(a + b)/2 - x| = |\breve{x} - (a + b)/2|$) and the interval's boundaries ($|x - a| = |b - \breve{x}|$) as well.

This definition can be extended to higher dimensions by applying the same formula to each dimension [Tizhoosh 2005b].

Definition 3.2 *Let $P(x_1, x_2, ..., x_D)$ be a point in D-dimensional space, where $x_1, x_2, ..., x_D$ are real numbers and $x_i \in [a_i, b_i]$, $i = 1, 2, ..., D$. The opposite point of P is denoted by $\breve{P}(\breve{x}_1, \breve{x}_2, ..., \breve{x}_D)$ where*

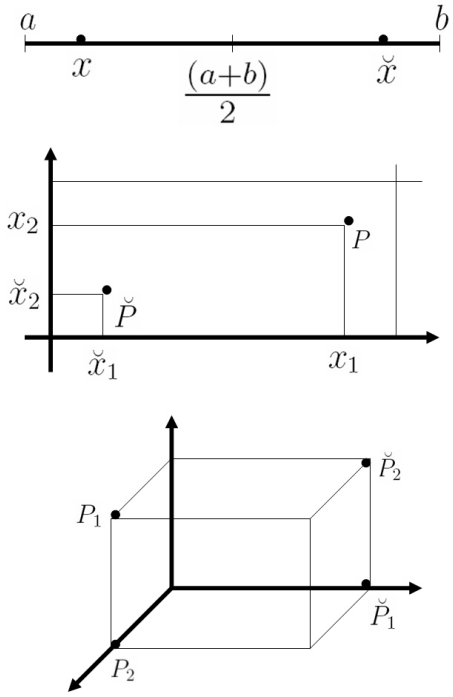


Fig. 3.1: Illustration of a point and its corresponding opposite in one, two, and three dimensional spaces.

$$\breve{x}_i = a_i + b_i - x_i. \tag{3.2}$$

Figure 3.1 illustrates a sample point and its corresponding opposite point in one, two, and three dimensional spaces.

3.2.2 Uniqueness

Theorem 3.1 (Uniqueness) *Every point* $P(x_1, x_2, ..., x_D)$ *in the D-dimensional space of real numbers with* $x_i \in [a_i, b_i]$ *has a unique opposite point* $\breve{P}(\breve{x}_1, \breve{x}_2, ..., \breve{x}_D)$ *defined by* $\breve{x}_i = a_i + b_i - x_i$, $i = 1, 2, 3, ..., D$.

Proof - Let us consider, without loss of generality, the two space corners a_1 and b_1 for the one dimensional case. According to the opposite point definition, we have $|x-a_1| = |\breve{x}-b_1|$ or $|\breve{x}-a_1| = |x-b_1|$. Now, assume that a second point x' is also opposite of x. Then, we should have $|x-a_1| = |x'-b_1|$ or $|x'-a_1| = |x-b_1|$. This, however, means $x' = \breve{x}$. Hence, $\breve{x}$ is unique.

3.3 Opposition-Based Optimization (OBO)

Now, after the definition of the opposite points we are ready to define *Opposition-Based Optimization (OBO).*

Definition 3.3 *Let $P(x_1, x_2, ..., x_D)$, a point in a D-dimensional space with $x_i \in [a_i, b_i]$ $(i = 1, 2, 3, ..., D)$, be a candidate solution. Assume $f(x)$ is a fitness function which is used to measure candidate optimality. According to opposite point definition, $\breve{P}(\breve{x}_1, \breve{x}_2, ..., \breve{x}_D)$ is the opposite of $P(x_1, x_2, ..., x_D)$. Now, if $f(\breve{P}) \geq f(P)$, then point P can be replaced with $\breve{P}$; otherwise we continue with P. Hence, the point and its opposite point are evaluated simultaneously to continue with the fitter one [Tizhoosh 2005b].*

This definition of OBO is making two fundamental assumptions. First, one of the candidate or the opposite candidate is always closer to the solution, and second, considering the opposition is more beneficial than generating additional random solutions and taking the best among them.

Empirical evidence for these claims will be provided in section 3.4. However, prior to providing experimental results, we also want to provide mathematical proofs.

3.3.1 Definitions, theorems, and proofs

Definition 3.4 *Euclidean distance between two points $P(p_1, p_2, ..., p_D)$ and $Q(q_1, q_2, ..., q_D)$ in a D-dimensional space is defined by*

$$d(P,Q) = \parallel P,Q \parallel = \sqrt{\sum_{i=1}^{D}(p_i - q_i)^2}. \tag{3.3}$$

It can be simplified as follows for a one-dimensional space:

$$d(P,Q) = \parallel P,Q \parallel = |p - q|. \tag{3.4}$$

Theorem 3.2 (First Opposition Theorem) *For any (unknown) function $y = f(x)$ ($x \in [a,b]$) with global optimum at x_s ($x_s \neq \frac{(a+b)}{2}$), the estimate solution x and its opposite $\breve{x}$, we have*

$$Pr\left(|\breve{x} - x_s| < |x - x_s|\right) = 1/2, \tag{3.5}$$

where $Pr(\cdot)$ is the probability function. It means candidate solution and its opposite have the equal chance to be closer to the solution.

Proof - We have $Pr(\breve{x} \in [a,(a+b)/2] | x > (a+b)/2) = 1$ and $Pr(\breve{x} \in [(a+b)/2,b] | x < (a+b)/2) = 1$. If $x_s \neq (a+b)/2$, then $Pr(|\breve{x} - x_s| < |x - x_s|) = 1/2$. For $x_s = (a+b)/2$, then $Pr(|\breve{x} - x_s| = |x - x_s|) = 1$.

Now, lets focus on the second assumption of OBO. Suppose the random variables $x_1, x_2, ...$ are continuous independent random variables representing the system inputs. Suppose the performance of the system for given inputs x_i is a monotone function $g(x_i)$. We wish to compare the performance of a system with independent inputs with one using opposition-based inputs. In particular, we wish to maximize some measure of performance $g(x)$ over possible inputs x. The following theorem shows the benefit of the opposite inputs, compared to random inputs.

Theorem 3.3 (Second Opposition Theorem) *For increasingly monotone g,*
$Pr\left(g(x_r) < max\{g(x), g(\breve{x})\}\right) = \frac{3}{4}$, where x is the first random guess; $\breve{x}$ is the opposite

point of x; and x_r is the second random guess.

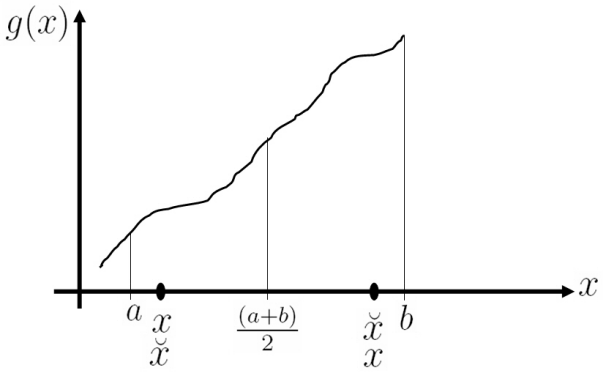


Fig. 3.2: Illustration of increasingly monotone g, interval boundaries, candidate and opposite candidate solutions.

Proof - Let us prove $Pr\left(g(x_r) > max\{g(x), g(\breve{x})\}\right) = 1 - Pr\left(g(x_r) < max\{g(x), g(\breve{x})\}\right) = \frac{1}{4}$ instead (see Figure 3.2),

$Pr\left(g(x_r) > max\{g(x), g(\breve{x})\}\right) = Pr\left(x < \frac{(a+b)}{2}\right) \times Pr\left(\breve{x} > \frac{(a+b)}{2}\right) \times Pr\left(x_r > \frac{(a+b)}{2}\right) \times Pr\left(x_r > \breve{x}\right) + Pr\left(x > \frac{(a+b)}{2}\right) \times Pr\left(\breve{x} < \frac{(a+b)}{2}\right) \times Pr\left(x_r > \frac{(a+b)}{2}\right) \times Pr(x_r > x) = \frac{1}{2} \times 1 \times \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times 1 \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{8} + \frac{1}{8} = \frac{1}{4}$.

Following results are obtained from this theorem:

1. $Pr\left(g(x) > g(x_r)\right) = \frac{3}{4} \times \frac{1}{2} = \frac{3}{8} = 0.375$
2. $Pr\left(g(\breve{x}) > g(x_r)\right) = \frac{3}{4} \times \frac{1}{2} = \frac{3}{8} = 0.375$
3. $Pr\left(g(x_r) > g(x) \wedge g(x_r) > g(\breve{x})\right) = Pr\left(g(x_r) > max\{g(x), g(\breve{x})\}\right) = \frac{1}{4} = 0.25$

Hence, by assuming g is a monotone function, the opposite point has 12.5% ($0.375 - 0.25 = 0.125$) higher chance to have a higher g value compared to the second random guess.

For the following central opposition theorems (one and D-dimensional), no assumption regarding the g function will be made.

Theorem 3.4 (Central Opposition Theorem for One-Dimensional Space)
Assume

(a) $y = f(x)$ $(x \in [a, b])$ *is an unknown function with at least one solution* $x_s \in [a, b]$ *for* $f(x) = \alpha$*; the solution can be anywhere in our search space (i.e., a black-box optimization problem),*

(b) x *is the first uniform random guess and* x_r *is the second uniform random guess in* $[a, b]$*; candidate solutions should be uniform random numbers because all points have the same chance to be the solution,*

(c) Opposite of $x \in [a, b]$ *is defined as* $\breve{x} = a + b - x$*,*

Then $Pr\left(|\breve{x} - x_s| < |x_r - x_s|\right) > Pr\left(|x_r - x_s| < |\breve{x} - x_s|\right)$.

In other words, the probability that the opposite point is closer to the solution is higher than probability of a second random guess.

Proof - In order to prove this theorem, we follow an exhaustive proof by covering all possible situations. Lets say, N different points over the interval $[a, b]$ divide it to $N + 1$ sub-intervals. So, three points $(x \neq (a + b)/2 \neq \breve{x})$ divide the interval $[a, b]$ to four sub-intervals $[a, x], [x, (a + b)/2], [(a + b)/2, \breve{x}]$, and $[\breve{x}, b]$. The solution x_s and the second random guess x_r can form 16 (4×4) different ways/combinations in the above mentioned four sub-intervals. Figure 3.3 illustrates all possible situations for a black-box optimization problem. We will call each situation an event. Therefore, we have 16 possible events (e_i, $i = 1, 2, ..., 16$). The probability of all events is equal because the solution (x_s), the first random guess (x), and the second random guess (x_r) can appear anywhere in the interval $[a, b]$ for a black-box optimization problem. Hence,

$$Pr(e_i) = \frac{1}{16}, \quad i = 1, 2, \ldots, 16. \tag{3.6}$$

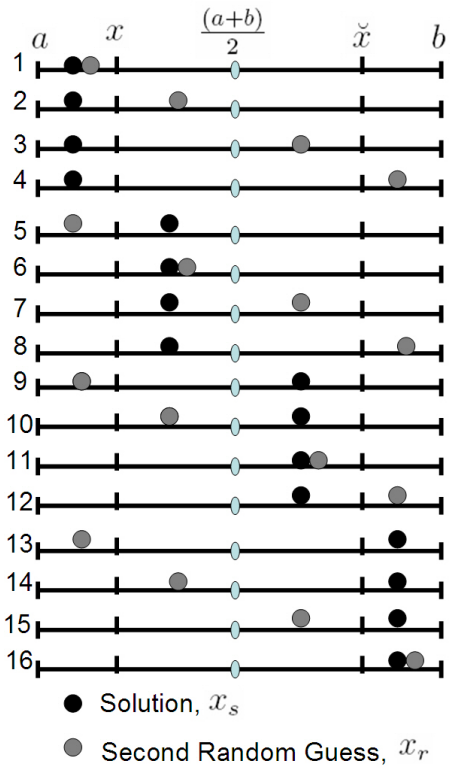


Fig. 3.3: All possible situations of x_r and x_s for a black-box optimization problem.

In order to establish an exhaustive proof, we start to calculate the following corresponding probabilities for each event:

p_x= probability of x being the closest to the solution x_s among $\{x, \breve{x}, x_r\}$,

p_r= probability of the second random guess x_r being the closest to the solution x_s among $\{x, \breve{x}, x_r\}$,

$p_{\breve{x}}$= probability of the opposite point $\breve{x}$ being the closest to the solution x_s among $\{x, \breve{x}, x_r\}$.

Obviously we have

$$p_x + p_{\breve{x}} + p_r = 1. \tag{3.7}$$

Now, all events are categorized into following four groups (see Figure 3.1):

1. $\text{Group}_1 = \{e_2, e_3, e_4, e_5, e_{12}, e_{13}, e_{14}, e_{15}\}$, $(x, \breve{x} \in [x_s, x_r])$.

   At least one of x or $\breve{x}$ is located between the x_s and x_r.

2. $\text{Group}_2 = \{e_8, e_9\}$, $(\min|x_s - x_r| \geq |x - x_s|) \vee (\min|x_s - x_r| \geq |\breve{x} - x_s|)$.

   Minimum distance between x_s and x_r is greater than distance between x_s and $x/\breve{x}$.

3. $\text{Group}_3 = \{e_7, e_{10}\}$, $(x_s \in [x, (a+b)/2]) \wedge (x_r \in [(a+b)/2, \breve{x}])$ or vice versa.

4. $\text{Group}_4 = \{e_1, e_6, e_{11}, e_{16}\}$, x_r and x_s are in the same interval.

In order to complete our table of probabilities step by step, the corresponding probabilities (p_x,p_r, and $p_{\breve{x}}$) are calculated for each group as follows:

Group$_1$: $\{e_2, e_3, e_4, e_5, e_{12}, e_{13}, e_{14}, e_{15}\}$

When $x \in [x_s, x_r]$ and it is closer to solution than $\breve{x}$, then x is clearly the closest to the solution (events: e_2, e_3, e_4, and e_5, Figure 3.1). Hence $p_x = 1$. Similarly, the same logic can be applied to $\breve{x}$ (events: e_{12}, e_{13}, e_{14}, and e_{15}, Figure 3.1). For these cases the entries can be inserted into the table of probabilities (Table 3.1). Newly added values are highlighted in boldface.

Group$_2$: $\{e_8, e_9\}$

(1) If $\min|x_s - x_r| \geq |x - x_s|$, so obviously $p_x = 1$, applicable to e_8.

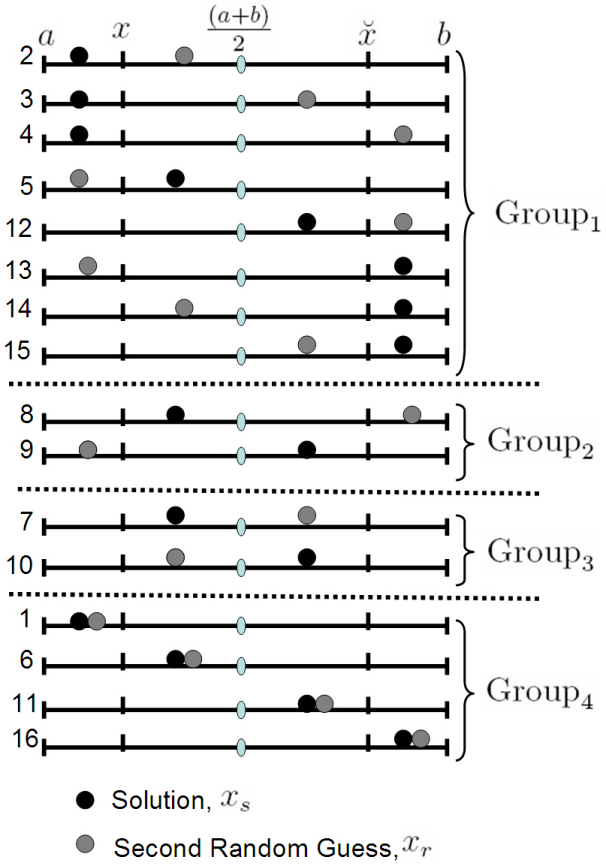


Fig. 3.4: Similar events are in the same group.

(2) Similarly, if $\min|x_s - x_r| \geq |\breve{x} - x_s|$, then obviously $p_{\breve{x}} = 1$, applicable to e_9.

These cases are completed in Table 3.1.

Group$_3$: $\{e_7, e_{10}\}$

Events e_7 and e_{10} are similar cases (Figure 3.1). Let us assume $p_x = \alpha$ for event e_7, so we have $p_r = 1 - \alpha$ because $p_x + p_{\breve{x}} + p_r = 1$ and $p_{\breve{x}} = 0$. Analogously, for event e_{10}, we have $p_{\breve{x}} = \alpha$, $p_r = 1 - \alpha$, and $p_x = 0$. We can complete our table for another two events (e_7 and e_{10}), see Table 3.1.

Table 3.1: Probabilities table after adding computations of Group_1, Group_2, and Group_3 (left to right, respectively).

event	p_{x_i}	p_{r_i}	$p_{\breve{x}_i}$	p_{x_i}	p_{r_i}	$p_{\breve{x}_i}$	p_{x_i}	p_{r_i}	$p_{\breve{x}_i}$
e_1	-	-	-	-	-	-	-	-	-
e_2	**1**	**0**	**0**	1	0	0	1	0	0
e_3	**1**	**0**	**0**	1	0	0	1	0	0
e_4	**1**	**0**	**0**	1	0	0	1	0	0
e_5	**1**	**0**	**0**	1	0	0	1	0	0
e_6	-	-	-	-	-	-	-	-	-
e_7	-	-	-	-	-	-	α	$(\mathbf{1}-\alpha)$	**0**
e_8	-	-	-	**1**	**0**	**0**	1	0	0
e_9	-	-	-	**0**	**0**	**1**	0	0	1
e_{10}	-	-	-	-	-	-	**0**	$(\mathbf{1}-\alpha)$	α
e_{11}	-	-	-	-	-	-	-	-	-
e_{12}	**0**	**0**	**1**	0	0	1	0	0	1
e_{13}	**0**	**0**	**1**	0	0	1	0	0	1
e_{14}	**0**	**0**	**1**	0	0	1	0	0	1
e_{15}	**0**	**0**	**1**	0	0	1	0	0	1
e_{16}	-	-	-	-	-	-	-	-	-

Now, let us have a preliminary estimate for α. Similar to the reason which was mentioned for Group_2 , we can conclude

$$\alpha > 1/2. \tag{3.8}$$

If $\min |x_r - x_s| \geq |x - x_s|$, so obviously $p_x = 1$, this case happens with a probability of at least $1/2$ when x_s is in the interval $[x, k]$ (the half of the interval $[x, (a+b)/2]$), see Figure 3.5.

Group_4: $\{e_1, e_6, e_{11}, e_{16}\}$

For this group, x_r and x_s are in the same interval. We just need to solve one of them.

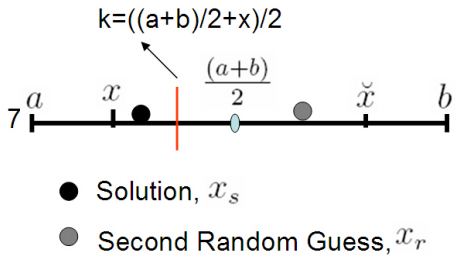


Fig. 3.5: Illustration of why the inequality $\alpha > 1/2$ holds. k is the center of the interval $[x, (a+b)/2]$.

Lets select event e_1; this case can be decomposed to four possible sub-cases, presented in Figure 3.6. For these sub-cases, the probabilities are given in Table 3.2. (note the recursive definition of the event (1b)). In order to calculate p_{1_x}, let

$$p_{1_x} = 1/4 \times p_{1_x} + 1/4 \times \alpha \Rightarrow p_{1_x} = \alpha/3. \tag{3.9}$$

We know $p(e_i) = 1/16$ (Eq. 3.6), so

$$p_{x_1} = 1/16 \times \alpha/3 \Rightarrow p_{1_x} = \alpha/48. \tag{3.10}$$

And finally

$$p_{r_1} = 1 - p_{x1} = 1 - \alpha/48 = (48 - \alpha)/48. \tag{3.11}$$

Now, we are ready to complete our table (see Table 3.3). According to our final probabilities table, we have

$$p_x = \sum_{i=1}^{16} p(e_i) p_{x_i} = (5 + 25\alpha/24)/16. \tag{3.12}$$

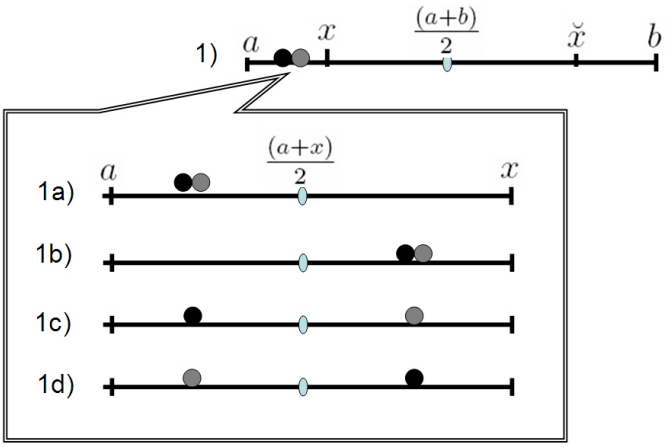


Fig. 3.6: Four possible sub-cases when x_r and x_s are in the same interval.

Table 3.2: Probabilities table of shown four cases in Figure 3.6.

event	$p_{1_{x_i}}$	$p_{1_{r_i}}$	$p_{1_{\breve{x}_i}}$
1a	0	1	0
1b	p_{1_x}	p_{1_r}	0
1c	0	1	0
1d	α	$(1-\alpha)$	0

$$p_r = \sum_{i=1}^{16} p(e_i) p_{r_i} = (6 - 50\alpha/24)/16. \tag{3.13}$$

$$p_{\breve{x}} = \sum_{i=1}^{16} p(e_i) p_{\breve{x}_i} = (5 + 25\alpha/24)/16. \tag{3.14}$$

Now, let us investigate when $p_{\breve{x}} > p_r$:

$$(p_{\breve{x}} > p_r) \Leftrightarrow \frac{5 + \frac{25\alpha}{24}}{16} > \frac{6 - \frac{50\alpha}{24}}{16} \tag{3.15}$$

Table 3.3: Probabilities table, final result.

event	p_{x_i}	p_{r_i}	$p_{\breve{x}_i}$
e_1	$(\alpha/48)$	$(48-\alpha)/48$	0
e_2	1	0	0
e_3	1	0	0
e_4	1	0	0
e_5	1	0	0
e_6	$(\alpha/48)$	$(48-\alpha)/48$	0
e_7	α	$(1-\alpha)$	0
e_8	1	0	0
e_9	0	0	1
e_{10}	0	$(1-\alpha)$	α
e_{11}	0	$(48-\alpha)/48$	$(\alpha/48)$
e_{12}	0	0	1
e_{13}	0	0	1
e_{14}	0	0	1
e_{15}	0	0	1
e_{16}	0	$(48-\alpha)/48$	$(\alpha/48)$
$(5+25\alpha/24)$,	$(6-50\alpha/24)$,		$(5+25\alpha/24)$

or

$$(p_{\breve{x}} > p_r) \Leftrightarrow \alpha > 24/75 \tag{3.16}$$

This is confirmed with $\alpha > 1/2$ (Eq. 3.8).

3.3.2 How much better is the opposite point?

Now we want to calculate an impact boundary for α and estimate the value of p_x, $p_{\breve{x}}$, and p_r. In the following two steps, we will find the lower and the upper boundaries for α.

Step 1. Calculating a lower boundary for α – Without loss of generality, we select e_7 to find the impact interval for α (Figure 3.7).

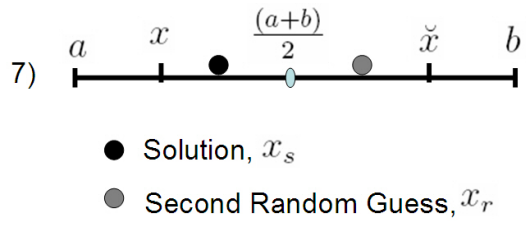


Fig. 3.7: The e_7 is selected to calculate an impact boundary for the α.

As mentioned before, if $\min|x_s - x_r| \geq |x - x_s|$, so obviously $p_x = 1$. As illustrated in Figure 3.8, we have

$$p_{x|_{min|x_s-x_r|\geq|x-x_s|}} = \lim_{N\to\infty} \sum_{i=1}^{N} P_{s_i} \times P_{r_i}. \tag{3.17}$$

P_{s_i} and P_{r_i} denote the probability of the presence of the solution and the second random guess in the shown intervals.

$$p_{x|_{min|x_s-x_r|\geq|x-x_s|}} = \lim_{N\to\infty} \left(\frac{1}{2^1} \times \frac{1}{2^0} + \frac{1}{2^2} \times \frac{1}{2^1} + \frac{1}{2^3} \times \frac{1}{2^2} + ... + \frac{1}{2^{(N+1)}} \times \frac{1}{2^N} \right) \tag{3.18}$$

$$p_{x|_{min|x_s-x_r|\geq|x-x_s|}} = \lim_{N\to\infty} \left(\frac{1}{2^1} + \frac{1}{2^3} + \frac{1}{2^5} + ... + \frac{1}{2^{(2N+1)}} \right) \tag{3.19}$$

This equation presents infinite geometric series; such series converge if and only if the absolute value of the common ratio is less than one ($|r| < 1$). For these kinds of series we have

$$\lim_{N\to\infty} \sum_{k=1}^{N} ar^k = \lim_{N\to\infty} \frac{a(1-r^{N+1})}{1-r} = \frac{a}{1-r}. \tag{3.20}$$

Hence

$$p_{x_{|min|x_s-x_r|\geq|x-x_s|}} = \frac{4}{6}. \tag{3.21}$$

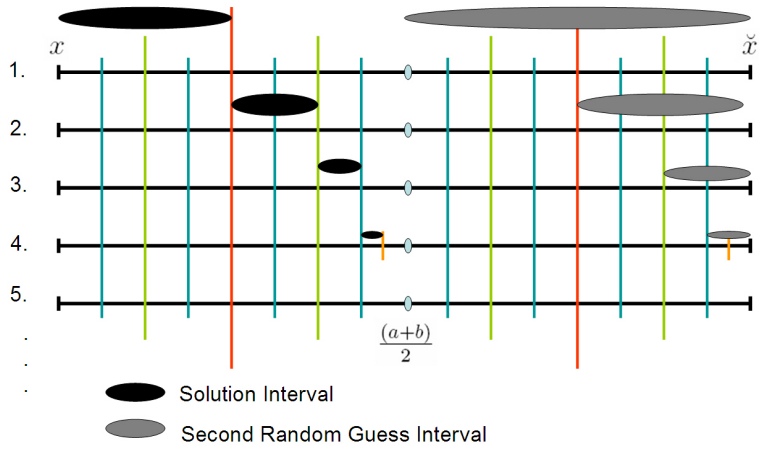


Fig. 3.8: Situations which $min|x_s - x_r| \geq |x - x_s|$.

So, we receive

$$\frac{4}{6} \leq \alpha. \tag{3.22}$$

Step 2. Calculating an upper boundary for α – Analogously, If $\min |x-x_s| \geq |x_s-x_r|$, then $p_r = 1$. So, as illustrated in Figure 3.9, we have

$$p_{r|_{min|x-x_s|\geq|x_s-x_r|}} = \lim_{N\to\infty} \sum_{i=1}^{N} P_{s_i} \times P_{r_i} \tag{3.23}$$

$$p_{r|_{min|x-x_s|\geq|x_s-x_r|}} = \lim_{N\to\infty} \left(\frac{1}{2^2} \times \frac{1}{2^1} + \frac{1}{2^3} \times \frac{1}{2^2} + \frac{1}{2^4} \times \frac{1}{2^3} + ... + \frac{1}{2^{(N)}} \times \frac{1}{2^{(N-1)}} \right) \tag{3.24}$$

$$p_{r|_{min|x-x_s|\geq|x_s-x_r|}} = \lim_{N\to\infty}\left(\frac{1}{2^3}+\frac{1}{2^5}+\frac{1}{2^7}+...+\frac{1}{2^{(2N-1)}}\right) \tag{3.25}$$

Again, we are faced with infinite geometric series and by reusing the Eq. 3.20, we have

$$p_{r|_{min|x-x_s|\geq|x_s-x_r|}} = \frac{1}{6}. \tag{3.26}$$

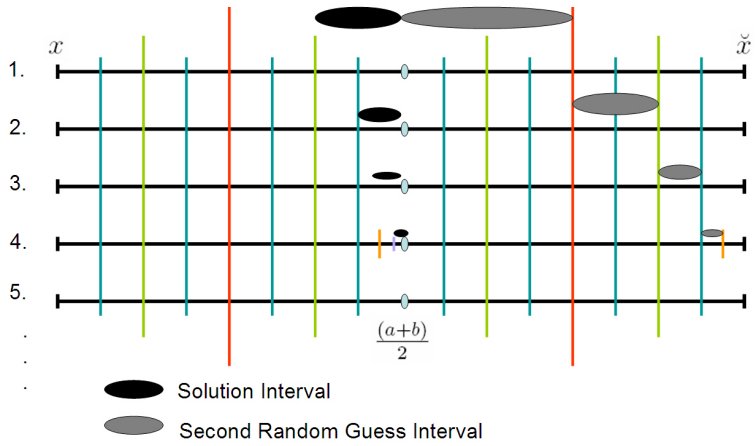


Fig. 3.9: Situations which $min|x-x_s| \geq |x_r-x_s|$.

Finally, we have

$$\alpha \leq (1-p_{r|_{min|x-x_s|\geq|x_s-x_r|}}) = \frac{5}{6}, \tag{3.27}$$

$$p_{x|_{min|x_s-x_r|\geq|x-x_s|}} \leq \alpha \leq (1-p_{r|_{min|x-x_s|\geq|x_s-x_r|}}), \tag{3.28}$$

or

$$\frac{4}{6} \leq \alpha \leq \frac{5}{6}. \tag{3.29}$$

By establishing this boundaries for α and considering Eq.s 3.12 - 3.14, we have

$$\frac{205}{576} \leq p_x = p_{\breve{x}} \leq \frac{845}{2304}. \tag{3.30}$$

or

$$0.36 \leq p_x = p_{\breve{x}} \leq 0.37. \tag{3.31}$$

And also

$$\frac{307}{1152} \leq p_r \leq \frac{83}{288}. \tag{3.32}$$

or

$$0.27 \leq p_r \leq 0.29. \tag{3.33}$$

Hence, the opposite of x ($\breve{x}$) has a higher chance to be closer to the solution, x_s, compared to the second random guess, x_r, in a one-dimensional solution space.

The center of the interval $[\frac{4}{6}, \frac{5}{6}]$ for α is $\frac{9}{12}$ or 0.75. By substituting this mean value in Eq.s 3.12 - 3.14, we receive

$$p_x = p_{\breve{x}} = 0.3613 \quad \text{and} \quad p_r = 0.2773. \tag{3.34}$$

Central opposition theorem can be extended to higher dimensions, following theorem addresses this extension.

Theorem 3.5 (Central Opposition Theorem for D-Dimensional Space)

Assume

(a) $y = f(X)$ *is an unknown function with* $X(x_1, x_2, x_3, ..., x_D)$, $x_i \in [a_i, b_i]$, $i = 1, 2, 3, ..., D$ *and at least one solution at* $X_s(x_{s_1}, x_{s_2}, x_{s_3}, ..., x_{s_D})$, $x_{s_i} \in [a_i, b_i]$, $i = 1, 2, 3, ..., D$,

(b) X is the first uniform random guess and $X_r(x_{r_1}, x_{r_2}, x_{r_3}, ..., x_{r_D})$ is the second uniform random guess in the solution space,

(c) The opposite point of $X(x_1, x_2, ..., x_D)$ is defined by $\breve{X}(\breve{x}_1, \breve{x}_2, ..., \breve{x}_D)$ where

$$\breve{x}_i = a_i + b_i - x_i, i = 1, 2, 3, ..., D.$$

Then $Pr\left(\|\breve{X}, X_s\| < \|X_r, X_s\|\right) > Pr\left(\|X_r, X_s\| < \|\breve{X}, X_s\|\right)$, where $\|\cdot\|$ denotes the Euclidean distance.

Proof - We have

$Pr\left(\|\breve{X}, X_s\| < \|X_r, X_s\|\right) > Pr\left(\|X_r, X_s\| < \|\breve{X}, X_s\|\right) =$

$$Pr\left(\sqrt{\sum_{i=1}^{D}(\breve{x}_i - x_{s_i})^2} < \sqrt{\sum_{i=1}^{D}(x_{r_i} - x_{s_i})^2}\right) > Pr\left(\sqrt{\sum_{i=1}^{D}(x_{r_i} - x_{s_i})^2} < \sqrt{\sum_{i=1}^{D}(\breve{x}_i - x_{s_i})^2}\right)$$

According to the Central Opposition Theorem for one-dimensional space we have

$$Pr\left(|\breve{x} - x_s| < |x_r - x_s|\right) > Pr\left(|x_r - x_s| < |\breve{x} - x_s|\right). \tag{3.35}$$

That is true for each dimension in the solution space, so

$$Pr\left(|\breve{x}_i - x_{s_i}| < |x_{r_i} - x_{s_i}|\right) > Pr\left(|x_{r_i} - x_{s_i}| < |\breve{x}_i - x_{s_i}|\right), i = 1, 2, 3, ..., D \tag{3.36}$$

Hence

$$Pr\left(\sqrt{\sum_{i=1}^{D}(\breve{x}_i - x_{s_i})^2} < \sqrt{\sum_{i=1}^{D}(x_{r_i} - x_{s_i})^2}\right) > Pr\left(\sqrt{\sum_{i=1}^{D}(x_{r_i} - x_{s_i})^2} < \sqrt{\sum_{i=1}^{D}(\breve{x}_i - x_{s_i})^2}\right),$$

and the Central Opposition Theorem is also valid for a D-dimensional space.

Table 3.4: Numerical results generated by Algorithm 2 (μ: Mean, σ: Standard deviation, CI: Confidence interval).

	p_x	$p_{\breve{x}}$	p_r
μ	0.3617	0.3617	0.2767
σ	0.0048	0.0048	0.0045
95% CI	$(0.3616, 0.3618)$	$(0.3616, 0.3618)$	$(0.2766, 0.2767)$

3.4 Empirical Verification

In this section, the aforementioned mathematical proofs are experimentally verified and the usefulness of the opposite numbers in higher dimensional spaces is investigated. For this propose, three random points in a D-dimensional space are generated (n times), called X, X_s, and X_r. Then, the number of times (out of n) which X, $\breve{X}$, or X_r is the closest to the randomly generated solution X_s (measured by Euclidean distance) is counted and finally the probability of the closeness of each point is calculated (p_x, $p_{\breve{x}}$, and p_r). In conducted experiments, n is chosen a large number in order to have an accurate estimation for probability values. The proposed method for this empirical verification is presented in Algorithm 2. As shown, there are two nested loops, the outer one to feed dimensions and the inner one to handle n trials for each dimension.

The experiments have been conducted for different dimensions ranging from D= 1 to D=10,000. In order to attain reliable results, the number of trials n was set to 1,000,000. Results are summarized in Table 3.4.

Results analysis - The mean μ (across all dimensions), standard deviation σ, and 95% confidence interval (CI) of p_x, $p_{\breve{x}}$, and p_r have been calculated for the results of dimensions D=$1, 2, 3, ..., 10,000$. Low standard deviations and short confidence intervals show that the probabilities remain the same for all investigated dimensions. The probability

Algorithm 2 Calculate p_x, $p_{\breve{x}}$, and p_r experimentally.

$D_{max} \leftarrow 10,000$
$n \leftarrow 1,000,000$
for $D = 1$ to D_{max} **do**
  for $R = 1$ to n **do**
    Generate three random points X, X_s, X_r in the D-dimensional space, $[a_i, b_i] = [-1, 1]$ for $i = 1, 2, 3, ..., D$
    Calculate the opposite point of X ($\breve{X}$)
    Calculate the Euclidean distance of X, $\breve{X}$, and X_r from X_s (d_X, $d_{\breve{X}}$, d_r)
    if $(d_X < d_{\breve{X}}) \wedge (d_X < d_r)$ **then**
      $c_x \leftarrow c_x + 1$ //x is the closest to the solution
    else if $(d_{\breve{X}} < d_X) \wedge (d_{\breve{X}} < d_r)$ **then**
      $c_{\breve{x}} \leftarrow c_{\breve{x}} + 1$ //$\breve{x}$ is the closest to the solution
    else if $(d_r < d_X) \wedge (d_r < d_{\breve{X}})$ **then**
      $c_r \leftarrow c_r + 1$ //x_r is the closest to the solution
    end if
  end for
  $p_x \leftarrow c_x/n$
  $p_{\breve{x}} \leftarrow c_{\breve{x}}/n$
  $p_r \leftarrow c_r/n$
end for

of opposite point $p_{\breve{x}}$ (0.3617) is 0.085 higher than the probability of a second random guess p_r (0.2767). As shown in Table 3.5, interestingly, experimental results conform with established theorems.

Additional experiments (not presented here) showed that $\alpha = 0.75$ is a proper value (a similar experimental method presented in this section is used to simulate e_7 and calculate α, see Figure 3.7). Using this empirical value in Eq.s 3.12 - 3.14, a more accurate comparison between theoretical and experimental probabilities can be provided (see Table 3.6). As seen, the probabilities are almost the same.

Table 3.5: Comparison of experimental and mathematical results.

	p_x	$p_{\breve{x}}$	p_r
Mathematical computation	$(0.3559, 0.3667)$	$(0.3559, 0.3667)$	$(0.2664, 0.2881)$
Experimental results	0.3617	0.3617	0.2767

Table 3.6: Comparison of experimental and mathematical results for $\alpha = 0.75$.

	p_x	$p_{\breve{x}}$	p_r
Mathematical computation ($\alpha = 0.75$)	0.3613	0.3613	0.2773
Experimental results	0.3617	0.3617	0.2767

3.5 Summary

This chapter established mathematical proofs and provided experimental evidence to verify the advantage of opposite points, compared to additional random points when dealing with high-dimensional problems. Both experimental and mathematical results conform with each other; opposite points are more beneficial than additional independent random points. Therefore we can conclude that the opposition-based optimization can be utilized to accelerate searching methods since considering the pair P and $\breve{P}$ has apparently a higher fitness probability than pure randomness.

Chapter 4

Opposition-Based Differential Evolution (ODE)

Therefore, the foundation of the creation was (based) upon opposites. Necessarily, we are battling because of loss and gain.

– Rumi (1207 – 1273) in "Masnawi"

4.1 Introduction

This chapter discusses how opposition-based optimization can be employed to accelerate population-based algorithms. In this direction, the opposition-based population initialization and generation jumping schemes are proposed. Simplicity and universality of the schemes are two main characteristics. The proposed schemes are utilized to introduce the opposition-based differential evolution (ODE). The acceleration of the DE algorithm is targeted by integration of the static opposite points in the initialization step and dynamic ones in the generation jumping. As it will be mentioned later, the considerable portion of the acceleration is obtained by generation jumping. The opposition-based initialization just provides fitter points for the start. The proposed schemes work at the population level and, for this reason, can be investigated for extension of other population-based algorithms as well.

4.2 Opposition-Based Acceleration Schemes

Generally speaking, in order to utilize the advantages of the opposition-based optimization to accelerate population-based algorithms, many schemes can be suggested and investigated. But, it seems that considering the following features during the design of these schemes are crucial:

Generality - Proposing general schemes makes it easy to use OBO for a wider range of population-based optimization methods. Tailored schemes would obviously be more rigid for generalization. Manipulating the internal operators of the optimizer leads to lower generality, although, the customized schemes can result a higher performance.

Simplicity - This feature is always desirable. Simplicity supports a higher understandability, and makes any design easy to implement and modify. Also, in practical environments, the simple schemes are widely appreciated.

Problem Independency - Proposed schemes have to be universal and capable to solve a wider range of optimization problems. By equipping the parent optimizer with the opposition-based schemes, it should not be specialized to solve a group of specific problems (e.g., unimodal). This case is experimentally verifiable by applying the algorithm to solve various global optimization problems. In other words, the proposed schemes should not reduce the universality of the parent optimizer to solve different problems.

Effectiveness - It should be taken into consideration that the evaluation of opposite points need more function calls and should be controlled smartly to prevent loosing the benefits through extra computations. Overall, the extra function calls should be reasonable and bring a benefit to the optimization process. The benefit can be faster convergence, higher robustness, or higher solution quality. Furthermore, improving one of these features should not affect the other benefits. During the experimental verification of the proposed algorithm, different measures are employed to investigate each criterion individually.

Any population-based optimization algorithm has two main phases, namely, population initialization and evolutionary generating of the new population; a general scheme is shown in Figure 4.1 (A). Three possible stages (marked by gray blocks in Figure 4.1 (B)) are recognizable to employ opposition-based optimization to accelerate the parent algorithm. These three stages are:

(1) During population initialization

(2) During population evolution, and

(3) After population evolution

As mentioned before, the evolutionary algorithms are categorized according to the employed evolutionary operators in their population evolution phase. So, any manipulation

in this phase reduces the generality of the proposed scheme. Exactly for this reason (to work at the population level), two stages, (1) and (3), are considered to employ OBO to accelerate the parent algorithm. Figure 4.2 presents the finalized general scheme for this proposal; through this way, two external blocks are selected and one internal block is removed. These two blocks are called opposition-based population initialization and opposition-based generation jumping, respectively. In the following subsections, the details about each block are provided.

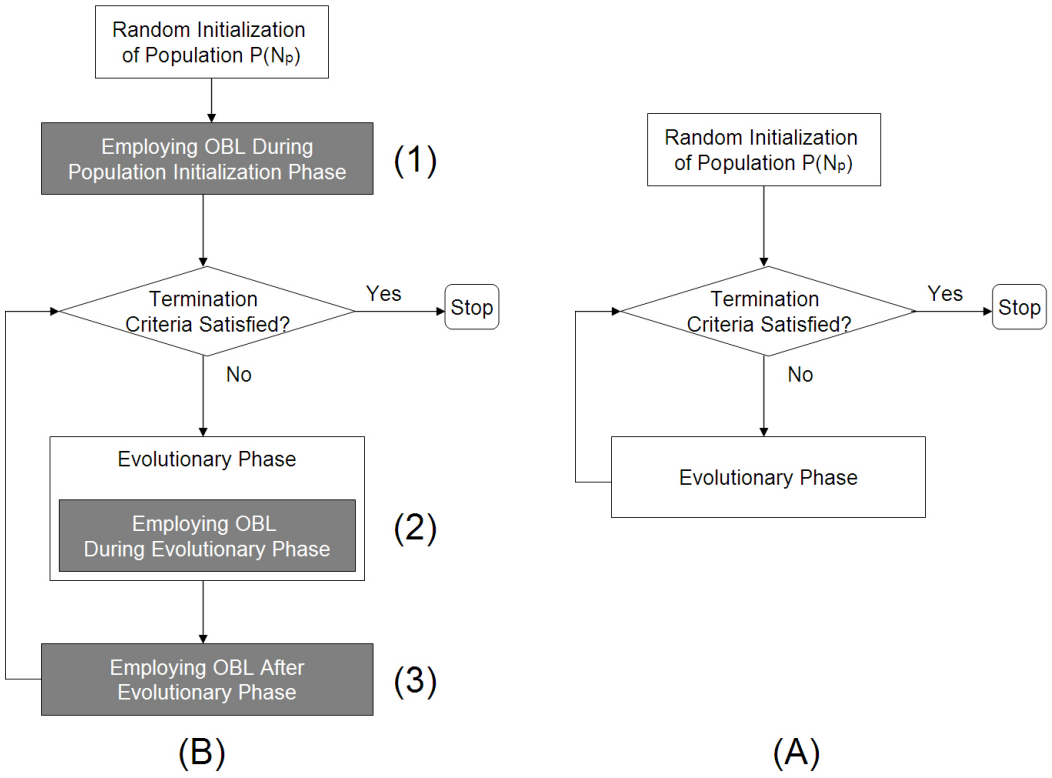


Fig. 4.1: (A) A general scheme for population-based optimization algorithms, (B) Three possible stages (marked by gray blocks) to employ OBO to accelerate (A).

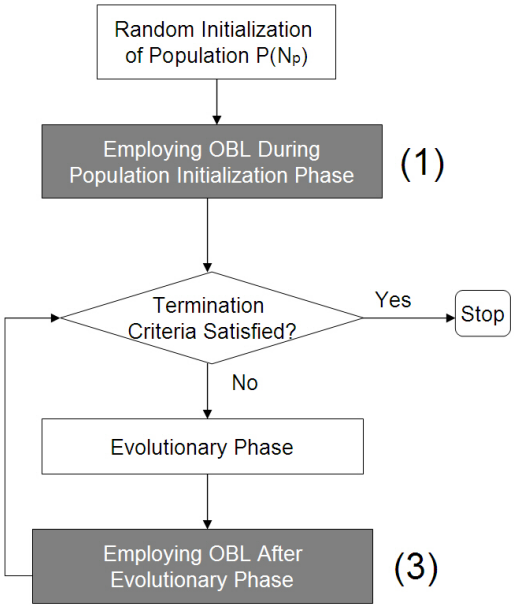


Fig. 4.2: In order to support generality for the opposition-based scheme two external blocks are chosen ,(1) and (3), to employ OBO. The internal one (2) was removed (see Figure 4.1 (B)).

4.2.1 Opposition-based population initialization

According to author's best knowledge, random number generation, in absence of a priori knowledge, is the commonly used method to create an initial population. But, as mentioned before, by utilizing OBO we can obtain fitter starting candidates even when there is no a priori knowledge about the solution(s). Block (1) from Figure 4.3 shows the implementation of opposition-based population initialization. Following steps explain that procedure:

step 1. Initialize the population $\mathrm{P}(N_p)$ randomly,

step 2. Calculate opposite population by

$$\mathrm{OP}_{i,j} = \mathrm{a}_j + \mathrm{b}_j - \mathrm{P}_{i,j}, \tag{4.1}$$

$$i = 1, 2, ..., N_p \ ; \ j = 1, 2, ..., D.$$

where $\mathrm{P}_{i,j}$ and $\mathrm{OP}_{i,j}$ denote the j^{th} variable of the i^{th} population and the opposite-population vector, respectively.

step 3. Select the N_p fittest individuals from the set $\{\mathrm{P} \cup \mathrm{OP}\}$ as the initial population.

According to the above procedure, $2N_p$ function evaluations are required instead of N_p for the regular random population initialization. But, by the opposition-based initialization, the parent algorithm can start with the fitter initial individuals instead, and this is a one-time cost.

4.2.2 Opposition-based generation jumping

By applying a similar approach to the current population, the evolutionary process can be forced to jump to a fitter generation. Based on a jumping rate J_r (i.e. jumping probability), after generating new populations by mutation, crossover, and selection, the opposite population is calculated and the N_p fittest individuals are selected from the union of the current population and the opposite population. As a difference to opposition-based initialization, it should be noted here that in order to calculate the opposite population for generation jumping, the opposite of each variable is calculated dynamically. That is, the maximum and minimum values of each variable in the *current population* ($[\mathrm{MIN}_j^p, \mathrm{MAX}_j^p]$) are used to calculate opposite points instead of using variables' predefined interval boundaries ($[a_j, b_j]$):

$$\mathrm{OP}_{i,j} = \mathrm{MIN}_j^p + \mathrm{MAX}_j^p - \mathrm{P}_{i,j}, i = 1, 2, ..., N_p; j = 1, 2, ..., D. \tag{4.2}$$

By staying within variables' static boundaries, it is possible to jump outside of the already shrunken search space and lose the knowledge of the current reduced space (converged population). Hence, we calculate opposite points by using variables' current interval in the population ($[\mathrm{MIN}_j^p, \mathrm{MAX}_j^p]$) which is, as the search does progress, increasingly

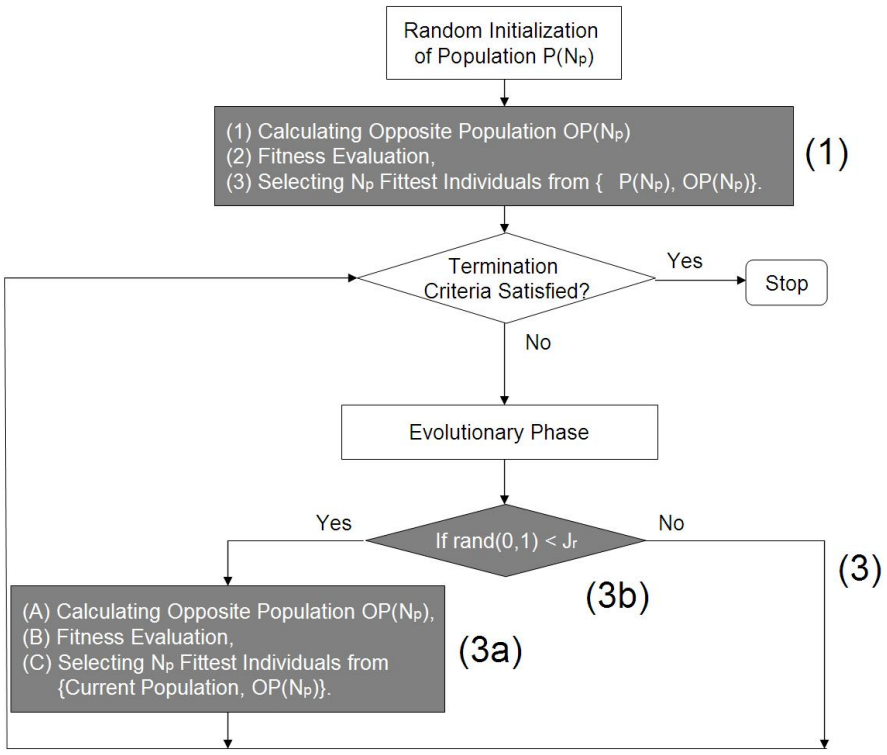


Fig. 4.3: New blocks are illustrated by gray boxes. Block (1): Opposition-based initialization, Block (3): Opposition-based generation jumping (J_r: jumping rate, $rand(0,1)$: uniformly generated random number, N_p: population size). Block (2) is removed.

smaller than the corresponding initial range $[a_j, b_j]$. Block (3) from Figure 4.3 indicates the implementation of opposition-based generation jumping.

A pictorial example is presented in Figure 4.4 to exhibit opposition-based generation jumping procedure in 2D space. 'S' indicates location of the solution. Dark and light circles present the points and the opposite points, respectively. As seen, in the resulted population (shown by the current P), the average distance of the selected candidates (which contains some original points and the opposite of some others) from the solution is smaller than it was for population (P) and opposite population (OP), individually.

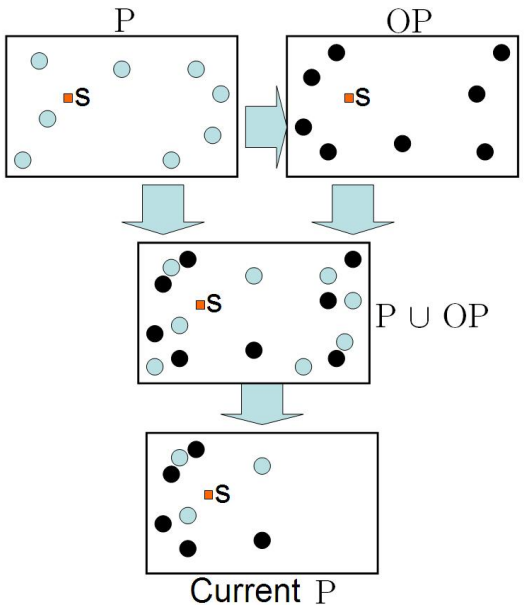


Fig. 4.4: A pictorial example to show the opposition-based generation jumping in 2D space ($N_p = 8$).

Furthermore, a real example for the opposition-based generation jumping in 2D space ($N_p = 1000$) is presented in Figure 4.5. The P, OP, $\{P \cup OP\}$, and the selected individuals after the population jumping are shown for some objective functions.

4.3 Opposition-Based Differential Evolution

Now, everything is ready to build opposition-based differential evolution (ODE). Similar to all population-based optimization algorithms, two main phases are distinguishable for the classical DE, namely, population initialization and producing new generations. DE is chosen as a parent algorithm for the proposed general scheme and the mentioned opposition-based population initialization and generation jumping are embedded inside DE, to increase the convergence speed. Corresponding flowchart and pseudo-code for the

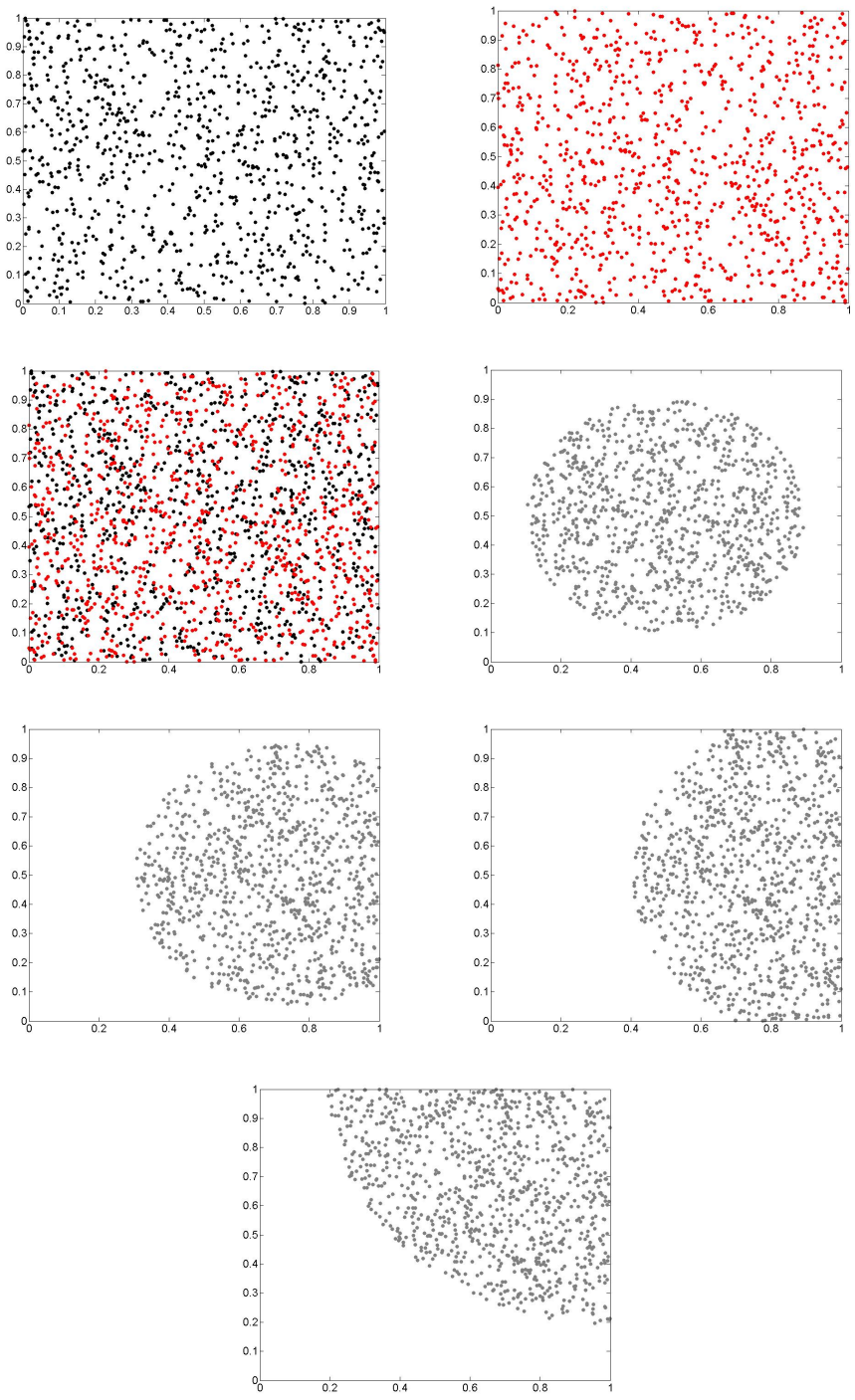


Fig. 4.5: A real example for the opposition-based generation jumping in 2D space ($N_p = 1000$). Top to bottom, left to right: P, OP, {P $\cup$ OP}, the selected individuals after the jumping for the following objective functions $f_1(x) = \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2}$, $f_2(x) = \sqrt{(x_1 - 0.75)^2 + (x_2 - 0.5)^2}$, $f_3(x) = \sqrt{(x_1 - 1)^2 + (x_2 - 0.5)^2}$, $f_4(x) = \sqrt{(x_1 - 1)^2 + (x_2 - 1)^2}$.

proposed approach (ODE) are given in Figure 4.6 and Algorithm 3, respectively. Block (1) from Figure 4.6 and steps 2-7 in Algorithm 3 show the implementation of opposition-based initialization and block (3) and steps 27-34 show the implementation of opposition-based generation jumping for ODE. The remaining tasks are DE's regular steps.

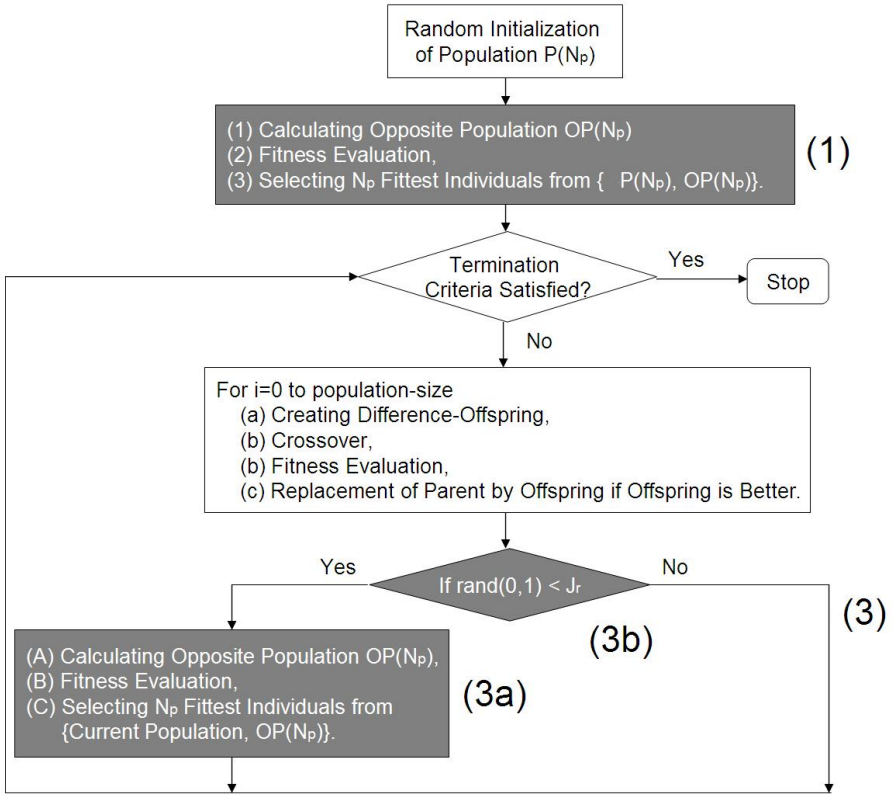


Fig. 4.6: Opposition-Based Differential Evolution (ODE).

Algorithm 3 Pseudo-code for Opposition-Based Differential Evolution (ODE). P_0: Initial population, OP_0: Opposite of initial population, P: Current population, OP: Opposite of current population, D: Problem dimension, $[a_j, b_j]$: Range of the j-th variable, J_r: Jumping rate, $\min_j^p/\max_j^p$: Minimum/maximum value of the j-th variable in the current population. Steps **2-7** and **27-34** are implementations of opposition-based population initialization and opposition-based generation jumping, respectively.

1: Generate uniformly distributed random population P_0
   //Begin of Opposition-Based Population Initialization
2: **for** $i = 0$ to N_p **do**
3: **for** $j = 0$ to D **do**
4: $OP_{0i,j} \leftarrow a_j + b_j - P_{0i,j}$
5: **end for**
6: **end for**
7: Select N_p fittest individuals from set the $\{P_0, OP_0\}$ as initial population P_0
   //End of Opposition-Based Population Initialization
   //Begin of DE's Evolution Steps
8: **while** (BFV > VTR **and** NFC < MAX_{NFC}) **do**
9: **for** $i = 0$ to N_p **do**
10: Select three parents P_{i_1}, P_{i_2}, and P_{i_3} randomly from current population where $i \neq i_1 \neq i_2 \neq i_3$
11: $V_i \leftarrow P_{i_1} + F \times (P_{i_2} - P_{i_3})$
12: **for** $j = 0$ to D **do**
13: **if** $rand(0,1) < C_r$ **then**
14: $U_{i,j} \leftarrow V_{i,j}$
15: **else**
16: $U_{i,j} \leftarrow P_{i,j}$
17: **end if**
18: **end for**
19: Evaluate U_i
20: **if** $(f(U_i) \leq f(P_i))$ **then**
21: $P'_i \leftarrow U_i$
22: **else**
23: $P'_i \leftarrow P_i$
24: **end if**
25: **end for**
26: $P \leftarrow P'$
   //End of DE's Evolution Steps
   //Begin of Opposition-Based Generation Jumping
27: **if** $rand(0,1) < J_r$ **then**
28: **for** $i = 0$ to N_p **do**
29: **for** $j = 0$ to D **do**
30: $OP_{i,j} \leftarrow \text{MIN}_j^p + \text{MAX}_j^p - P_{i,j}$
31: **end for**
32: **end for**
33: Select N_p fittest individuals from set the $\{P, OP\}$ as current population P
34: **end if**
   //End of Opposition-Based Generation Jumping
35: **end while**

4.4 Summary

A general opposition-based scheme for population-based algorithms is proposed in this chapter. The details for opposition-based population initialization and generation jumping are provided, and, these two new blocks are embedded inside classical DE to introduce the opposition-based DE. In the proposed scheme, evolutionary part of the algorithm is kept untouched to support a higher generality. The rate of generation jumping can be controlled by the jumping rate (J_r) and each block (initialization and generation jumping) works independently. Opposition-based population initialization provides fitter individuals to start; and the opposition-based generation jumping forces the population to jump and continue with a fitter generation. Both newly embedded blocks should accelerate the convergence rate of the DE. Experimental results will support this expectation. Discussion about the control parameter (J_r) will be provided later.

Chapter 5

Empirical Study and Analysis

The strongest arguments prove nothing so long as the conclusions are not verified by experience. Experimental science is the queen of sciences and the goal of all speculation.

– Roger Bacon (c. 1214–1294)

5.1 Introduction

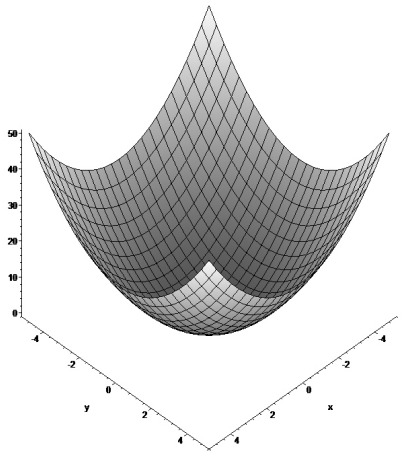
Similar to other evolutionary algorithms and due to stochastic nature, a strong convergence proof for differential evolution does not exist [Feoktistov 2006]. It means that even two different versions of DE cannot be compared mathematically, such that experimental comparison is required (unlike deterministic algorithms which can be compared through algorithm complexity analysis). Like many other studies in this field, a comprehensive benchmark test suite is employed to empirically analyze the ODE. Nine experimental series have been designed and conducted. In subsections 5.5.7-5.5.9, ODE is compared with six other evolutionary algorithms. Convergence speed, success rate, and solution quality are three core measures in the following studies.

5.2 Benchmark Test Suite

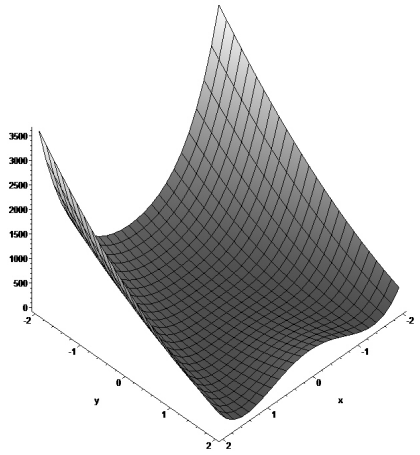
A comprehensive set of benchmark functions, including 58 different global optimization problems, has been employed for performance verification of the proposed approach (although utilizing a much smaller set of benchmark functions for this purpose is commonly acceptable, e.g. [Koumousis and Katsaras 2006]). The definition of the benchmark functions and their global optimum(s) are listed in Appendix A. Generally, following characteristics are desirable to provide a comprehensive test suite:

Selecting well-known test functions: All 58 functions are frequently used benchmark problems in global optimization field. However, there is no unique standard test set; researchers usually use a subset of well-known functions to validate their work. Generally, the size of the test suite is between 5 and 25 functions.

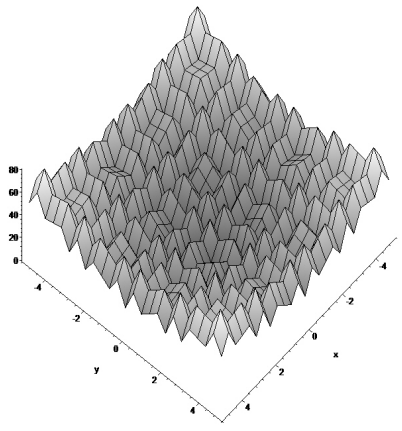
Some sample 3-D maps for 2-D functions from the selected test suite are presented in Figures 5.1 and 5.2.



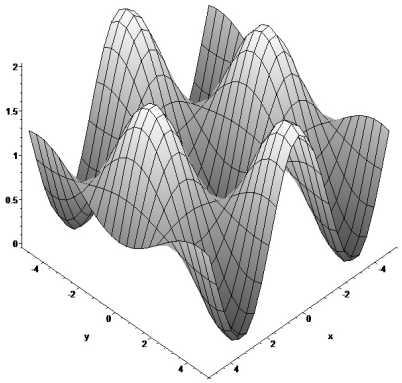
(a) f_1 (1^{st} De Jong) is unimodal, scalable, convex, and easy function.



(b) f_4 (Rosenbrock's Valley) is unimodal, scalable, non-convex, and hard function. The minimum is inside a long, narrow, parabolic shaped flat valley

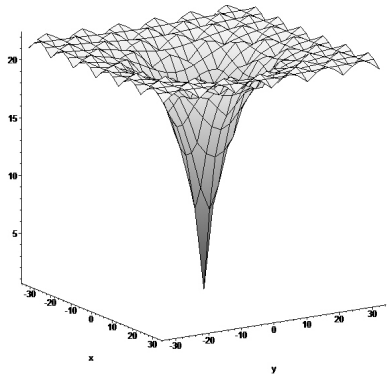


(c) f_5 (Rastrigin's Function) is highly multimodal. The location of the minima are regularly distributed.

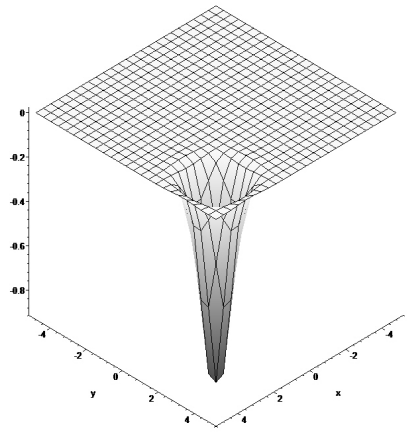


(d) f_6 (Griewangk's Function) has many regularly distributed local minima and hard to locate global minimum.

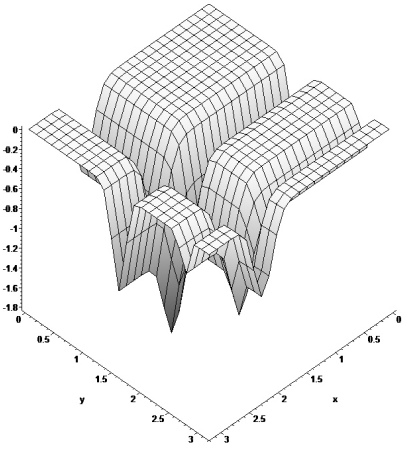
Fig. 5.1: Some sample 3-D maps for 2-D functions selected from the test suite.



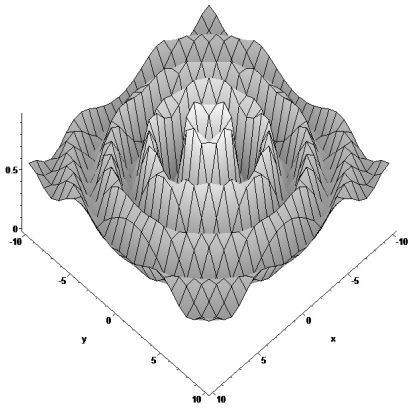
(a) f_8 (Ackley's Problem), the number of local minima is unknown.



(b) f_{11} (Easom Function) is unimodal and the global minimum has a small area relative to the search space.



(c) f_{18} (Michalewicz Function) has $n!$ local minima. Steepness of the valleys or edges makes it a hard optimization problem.



(d) f_{32} (Schaffer's Function 6) is multimodal and symmetric.

Fig. 5.2: Continued from Figure 5.1.

Modality: With a simple, but not exact definition, all benchmark functions can be categorized in three groups:

1) Unimodal functions (functions with one global optimum, e.g., Figures 5.1(a), 5.1(b), and 5.2(b)),
2) Multimodal functions with a few local optima (e.g., Figure 5.1(d)),
3) Multimodal functions with many local optima (e.g., Figures 5.1(c), 5.2(a), 5.2(c), and 5.2(d)).

In general, escaping from local optima to find the global optimum is a challenging task for any optimizer. Generally speaking, the multimodal functions are harder than unimodal ones, but the functions' shape can affect the hardness of the problem even for a unimodal function. As an example, Rosenbrock's Valley (Figure 5.1(b)) and Easom Function (Figure 5.2(b)) are unimodal functions, but both are difficult; for the first one the minimum is inside a long, narrow, parabolic shaped flat valley and for the second one the global minimum has a small area relative to the search space (small basin).

In our test suite, 23 of functions are unimodal, 18 are multimodal with a few (smaller than 10) local optima, and 17 are highly multimodal functions.

Scalability: Possessing 20 scalable functions in our test set provides the opportunity to validate the proposed approach for different dimensions. On the other hand, scalable functions are usually separable (means the parameters can be optimized individually). In revers manner, non-scalable functions tend to be nonseparable. Majority of the functions in our test suite are nonseparable functions, which are challenging for any optimization contest.

Wide dimension size: The dimensions of the test functions are distributed mostly between 2 and 60. In the conducted dimensionality study, 49 low dimensional func-

tions ($D \leq 10$) and 49 high dimensional functions ($D > 10$) have been tested.

Including eccentric solution problems: Sometimes, algorithms are biased toward the center of the searching space. Hence, they perform better with the functions which the solution is in the center of the search space. To prevent biased results, 33 of the functions have the optimal solutions not located in the center of search space.

Complex structures: Existence of small attraction basins, saddle points, steep edges, padding with noise, distributed local optima around the global optimum, and lack of useful information on function's global structure (e.g., flat structure) are some examples which make optimization much more challenging. For each mentioned source of complexity, there is at least one representative in the test suite.

5.3 Comparison Strategies and Metrics

We compare the convergence speed of DE and ODE by measuring the *number of function calls* (NFC) which is the most commonly used metric in literature [Price et al. 2005; Vesterstroem and Thomsen 2004; Hrstka and Kučerová 2004; Suganthan et al. 2005]. A smaller NFC means higher convergence speed. The termination criterion is to find a value smaller than the value-to-reach (VTR) before reaching the maximum number of function calls $\mathrm{MAX_{NFC}}$. VTR is selected as a very small value (e.g. 10^{-8}) to guarantee high accuracy. In order to minimize the effect of the stochastic nature of the algorithms on the metric, the reported number of function calls (NFC) for each function is the average over 50 trials. In order to compare convergence speeds, we use the acceleration rate (AR) which is defined as follows, based on the number of function calls for the two algorithms DE and ODE:

$$\mathrm{AR} = \frac{\mathrm{NFC_{DE}}}{\mathrm{NFC_{ODE}}}, \tag{5.1}$$

where AR > 1 means ODE is faster.

The number of times, for which the algorithm succeeds to reach the VTR for each test function is measured as the success rate (SR) [Suganthan et al. 2005]:

$$\text{SR} = \frac{\text{number of times reached VTR}}{\text{total number of trials}}. \tag{5.2}$$

SR is a commonly used metric to quantify the robustness of the algorithms.

Further, the *average acceleration rate* (AR_{ave}) and the *average success rate* (SR_{ave}) over n test functions are calculated as follows:

$$\text{AR}_{ave} = \frac{1}{n}\sum_{i=1}^{n}\text{AR}_i, \tag{5.3}$$

$$\text{SR}_{ave} = \frac{1}{n}\sum_{i=1}^{n}\text{SR}_i. \tag{5.4}$$

These two average metrics help us to have an overall comparison for the entire test suite; other measures such as number of solved ($SR \neq 0$) problems and number of problems for which the algorithm shows better results than other competitor(s) are taken to account for this purpose.

5.4 Parameter Settings

Parameter settings for all conducted experiments are as follows unless a change is mentioned (the same setting has been used in literature cited after of each parameter):

- Population size, $N_p = 100$ [Brest et al. 2006; Yao et al. 1999; Lee and Yao 2004]
- Differential amplification factor, $F = 0.5$ [Storn and Price 1997a; Ali and Törn 2004; Liu and Lampinen 2005; Brest et al. 2006; Rahnamayan and Dieras 2007]

- Crossover probability constant, $C_r = 0.9$ [Storn and Price 1997a; Ali and Törn 2004; Liu and Lampinen 2005; Brest et al. 2006; Rahnamayan and Dieras 2007]
- Jumping rate constant, $J_r = 0.3$ (discussed in subsection 5.5.6)
- Mutation strategy: DE/rand/1/bin (classic version of DE) [Storn and Price 1997a; Price et al. 2005; Onwubolu and Babu 2004; Brest et al. 2006; Sun et al. 2005]
- Maximum number of function calls, $\text{MAX}_{\text{NFC}} = 10^6$ [Rahnamayan et al. 2006f]
- Value to reach, VTR= 10^{-8} [Suganthan et al. 2005]

In order to maintain a reliable and fair comparison, (a) the parameter settings are the same as above for all experiments, unless we mention new settings to serve the purpose of that parameter study, (b) for all conducted experiments, the reported values are the average of the results for 50 independent runs, and the last and also more important one, (c) needless to say, extra fitness evaluations required for the opposite points (both in population initialization and also generation jumping phases) are counted as well.

5.5 Experimental Studies

A comprehensive set of experiments has been conducted and they are categorized as follows. In subsection 5.5.1, DE and ODE are compared in terms of convergence speed and robustness. The effect of problem dimensionality is investigated in subsection 5.5.2. The contribution of opposite points to the achieved acceleration results is demonstrated in subsection 5.5.3. The effect of population size is studied in subsection 5.5.4. Comparison of DE and ODE over different mutation operators is performed in subsection 5.5.5. Discussion about the control parameter, jumping rate, is covered in subsection 5.5.6. And finally, ODE is compared with DE, Fuzzy Adaptive DE (FADE), Adaptive LEP, Best Lévy,

Fast Evolutionary Programming (FEP), and Classical Evolutionary Programming (CEP) in subsections 5.5.7-5.5.9.

5.5.1 Experiment series 1: comparison of DE and ODE

First of all, we need to compare the parent algorithm, DE, with ODE in terms of convergence speed and robustness. The results for solving 58 benchmark functions (see Appendix A) are given in Table 5.1. The best result for the number of function calls (NFC) and the success rate (SR) for each function are highlighted in boldface. The average success rates (SR_{ave}) and the average acceleration rate (AR_{ave}) on 58 test functions are shown in the last row of the table.

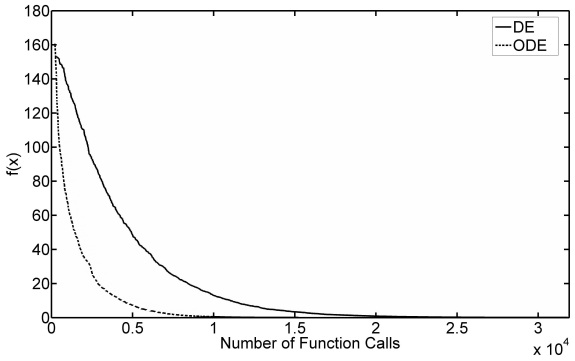
ODE outperforms DE on 40 test functions (69% of problems) while DE surpasses ODE on 15 functions (26% of problems). Over the remaining 3 functions (5% of cases) they perform the same. Except for f_4, the rest of 14 functions are low-dimensional functions ($D \leq 10$). Average acceleration rate (AR_{ave}) is 1.44 which means ODE is on average 44% faster than DE. While the average success rate (SR_{ave}) for both is equal to 0.86, both algorithms fail to solve f_{13}, f_{26}, and f_{27}; in addition, DE fails to solve f_{51} and ODE is unsuccessful on f_4. Some sample graphs for the performance comparison between DE and ODE are given in Figure 5.3. These curves (best solution vs. number of function calls) show that ODE converges faster than DE toward the optimal solution.

Results analysis - With the same control parameter settings for both algorithms and fixing the jumping rate for the ODE ($J_r = 0.3$), their success rates are comparable while ODE shows better convergence speed than DE (44% faster overall). Jumping rate is an important control parameter which, if optimally set, can achieve even better results; the discussion about this parameter is covered in subsection 5.5.6.

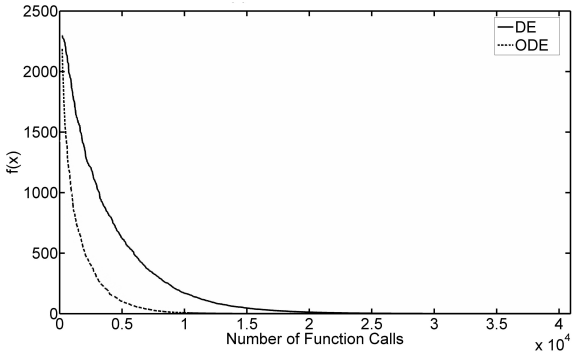
Table 5.1: Comparison of DE and ODE. D: Dimension, NFC: Number of function calls, SR: Success rate, AR: Acceleration rate. The last row of the table presents the average success rates (SR_{ave}) and the average acceleration rate (AR_{ave}). The best results are highlighted in **boldface**. AR> 1 means ODE performs faster.

		DE		*ODE*					*DE*		*ODE*		
F	*D*	*NFC*	*SR*	*NFC*	*SR*	*AR*	*F*	*D*	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_1	30	87748	1	**47716**	1	1.83	f_{30}	2	1016	1	**996**	1	1.02
f_2	30	96488	1	**53304**	1	1.81	f_{31}	30	411164	1	**337532**	1	1.22
f_3	20	177880	1	**168680**	1	1.05	f_{32}	2	7976	1	**5092**	1	1.56
f_4	30	**403112**	**1**	–	0	–	f_{33}	5	2163	0.88	**2024**	**1**	1.07
f_5	10	328844	**1**	**70389**	0.76	4.67	f_{34}	5	38532	1	**16340**	1	2.36
f_6	30	113428	**1**	**69342**	0.96	1.64	f_{35}	2	2052	1	**1856**	1	1.11
f_7	30	25140	1	**8328**	1	3.01	f_{36}	2	8412	1	**5772**	1	1.46
f_8	30	169152	1	**98296**	1	1.72	f_{37}	2	5284	1	**4728**	1	1.12
f_9	2	**4324**	1	4776	1	0.90	f_{38}	2	5280	1	**4804**	1	1.10
f_{10}	4	**16600**	1	19144	1	0.87	f_{39}	2	3780	1	**3396**	1	1.11
f_{11}	2	8016	1	**6608**	1	1.21	f_{40}	2	2424	1	**2152**	1	1.13
f_{12}	3	**3376**	1	3580	1	0.94	f_{41}	10	19528	1	**15704**	1	1.24
f_{13}	6	–	0	–	0	–	f_{42}	2	4780	1	**4684**	1	1.02
f_{14}	2	5352	1	**4468**	1	1.20	f_{43}	3	**6852**	1	8484	1	0.81
f_{15}	30	101460	1	**70408**	1	1.44	f_{44}	3	7036	1	**6172**	1	1.14
f_{16}	100	3608	1	**3288**	1	1.09	f_{45}	2	3256	1	**3120**	1	1.04
f_{17}	4	549850	0.04	**311800**	**0.12**	1.76	f_{46}	3	6184	1	**5472**	1	1.13
f_{18}	10	**191340**	**0.76**	213330	0.56	0.90	f_{47}	2	2976	1	**2872**	1	1.03
f_{19}	30	385192	1	**369104**	1	1.04	f_{48}	4	**1108**	1	1232	1	0.90
f_{20}	2	**4884**	1	5748	1	0.85	f_{49}	2	**5232**	**1**	5956	0.92	0.88
f_{21}	30	187300	1	**155636**	1	1.20	f_{50}	4	379900	**1**	**250260**	0.20	1.52
f_{22}	30	570290	0.28	**72250**	**0.88**	7.89	f_{51}	10	–	0	**16681**	**0.84**	–
f_{23}	30	41588	1	**23124**	1	1.80	f_{52}	10	**14968**	1	15104	1	0.99
f_{24}	30	818425	0.12	**470359**	**1**	1.78	f_{53}	2	7888	1	**4272**	1	1.85
f_{25}	4	**13925**	**0.80**	15280	0.60	0.91	f_{54}	4	8856	1	**7504**	1	1.18
f_{26}	4	–	0	–	0	–	f_{55}	9	**78567**	0.24	97536	**0.56**	0.81
f_{27}	4	–	0	–	0	–	f_{56}	10	37824	1	**24260**	1	1.56
f_{28}	4	4576	1	**4500**	1	1.02	f_{57}	2	**30704**	**1**	55980	0.76	0.55
f_{29}	2	**10788**	1	11148	1	0.97	f_{58}	4	**16600**	1	19144	1	0.87

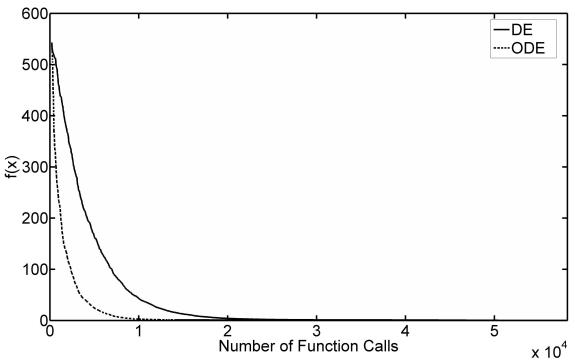
$\mathrm{SR}_{ave}(DE) = 0.86, \mathrm{SR}_{ave}(ODE) = 0.86, \mathrm{AR}_{ave} = 1.44$



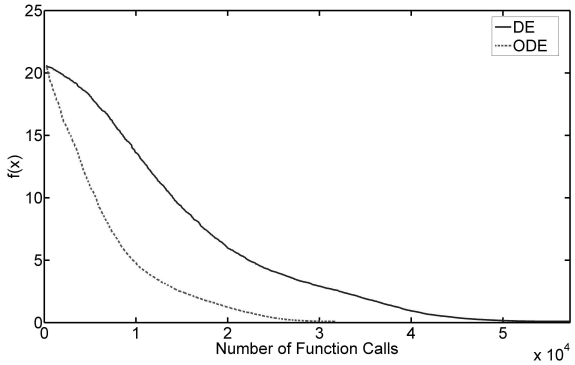
(a) f_1, ODE is 1.83 times faster.



(b) f_2, ODE is 1.81 times faster.



(c) f_6, ODE is 1.64 times faster.



(d) f_8, ODE is 1.72 times faster.

Fig. 5.3: Sample convergence graphs (best solution vs. number of function calls).

5.5.2 Experiment series 2: influence of dimensionality

In order to investigate the effect of the problem dimensionality, the same experiments are repeated for D′=D/2 and D′=2D for each *scalable function* in our test set. All control parameters remain unchanged. Results for D/2 and 2D are illustrated in Table 5.2 for 40 test functions.

According to the obtained results, ODE surpasses DE on 34 test functions while DE

outperforms ODE on 2 functions ($f_{4(D=15)}$ and $f_{18(D=5)}$). Both algorithms are unable to solve $f_{4(D=60)}$, $f_{19(D=60)}$, $f_{22(D=60)}$, and $f_{51(D=20)}$ before meeting the maximum number of function calls. The average acceleration rate is equal to 1.73, meaning that ODE performs 73% faster than DE. The average success rate for DE and ODE are 0.82 and 0.81, respectively.

For 11 functions ($f_1, f_2, f_6, f_7, f_8, f_{15}, f_{16}, f_{18}, f_{21}, f_{41}$, and f_{56}), the acceleration rate (AR) increases as the dimensionality grows. ODE achieves more desirable results for all but 4 of the functions (f_3, f_5, f_{23}, and f_{31}) where no improvement can be observed. An interesting effect for f_{18} is that for dimensions 5 and 10 (for D= 10 see Table 5.1), DE performs better than ODE; but when the dimension is increased to 20, ODE shows better results in terms of NFC and SR. Furthermore, DE cannot solve f_{24} for D= 60, but ODE solves it in 35% of the trials.

At the bottom of the Table 5.2, the average success rates and the average acceleration rates for functions with D/2, 2D are presented. For functions with dimension of D/2, the overall success rate for DE is 4% higher than ODE's (0.98 vs. 0.94) and the overall acceleration rate is 1.67. For functions with dimension of 2D, overall success rate of DE and ODE are almost the same (0.66 vs. 0.67, respectively). But this value is smaller than what we had observed for D/2. For this case the overall acceleration rate is 1.81.

Results analysis - Decreasing of the overall success rate for DE and ODE was predictable because if we double the problem dimension, algorithms are sometimes unable to solve the problem before reaching the maximum number of function calls (which is a fixed number for all experiments). But as seen, ODE performs better for high dimensional problems. The higher average acceleration rate has been achieved for functions with dimension 2D.

Table 5.2: Comparison of DE and ODE for dimension sizes D/2 and 2D for all scalable functions of the test suite. At the bottom of the table, the average success rates and the average acceleration rates for functions with D/2, 2D, and for both (overall) are presented. The best values of NFC and SR for each case are highlighted in boldface.

		DE		*ODE*					*DE*		*ODE*		
F	*D*	*NFC*	*SR*	*NFC*	*SR*	*AR*	*F*	*D*	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_1	15	39920	1	**27860**	1	1.43	f_{18}	5	**18575**	0.96	20067	0.96	0.93
	60	156552	1	**93720**	1	1.67		20	288300	0.35	**253910**	**0.55**	1.14
f_2	15	43268	1	**30956**	1	1.40	f_{19}	15	87856	1	**86372**	1	1.02
	60	178308	1	**108960**	1	1.64		60	–	0	–	0	–
f_3	10	49188	1	**45284**	1	1.09	f_{21}	15	83324	1	**78916**	1	1.06
	40	835316	1	**795476**	1	1.05		60	330172	1	**246552**	1	1.34
f_4	15	**135452**	**1**	–	0	–	f_{22}	15	129276	1	**51932**	1	2.49
	60	–	0	–	0	–		60	–	0	–	0	–
f_5	5	46964	**1**	**20252**	1	2.32	f_{23}	15	18248	1	**13252**	1	1.38
	20	817000	0.08	**412240**	**0.16**	1.98		60	73756	1	**56708**	1	1.30
f_6	15	108360	**0.96**	**75018**	0.88	1.44	f_{24}	15	372000	1	**258650**	1	1.44
	60	194612	**1**	**127490**	0.72	1.53		60	–	0	**795575**	**0.35**	–
f_7	15	13776	1	**7264**	1	1.90	f_{31}	15	265872	1	**216880**	1	1.23
	60	45276	1	**8620**	1	5.25		60	440320	1	**398956**	1	1.10
f_8	15	80520	1	**58300**	1	1.38	f_{41}	5	8060	1	**7220**	1	1.12
	60	295172	1	**188332**	1	1.57		20	46216	1	**27736**	1	1.55
f_{15}	15	45726	1	**39236**	1	1.17	f_{51}	5	45237	0.76	**6776**	**1**	6.68
	60	180260	**0.84**	**121750**	0.60	1.48		20	–	0	–	0	–
f_{16}	50	3628	1	**3296**	1	1.10	f_{56}	5	11524	1	**9848**	1	1.17
	200	3680	1	**3264**	1	1.13		20	179040	1	**55616**	1	3.22
D/2, $\mathrm{SR}_{ave}(DE) = 0.98, \mathrm{SR}_{ave}(ODE) = 0.94, \mathrm{AR}_{ave} = 1.67$													
2D, $\mathrm{SR}_{ave}(DE) = 0.66, \mathrm{SR}_{ave}(ODE) = 0.67, \mathrm{AR}_{ave} = 1.81$													
Overall, $\mathrm{SR}_{ave}(DE) = 0.82, \mathrm{SR}_{ave}(ODE) = 0.81, \mathrm{AR}_{ave} = 1.73$													

5.5.3 Experiment series 3: contribution of opposite points

In this section, we want to verify that the achieved acceleration rate is really due to utilizing opposite points and not due to additional sampling. For this purpose, all parts of the proposed algorithm are kept untouched and instead of using opposite points for the population initialization and the generation jumping, uniformly generated random points are employed. In order to have a fair competition for this case, exactly like what we underwent for opposite points, the predefined boundaries of variables ($[a_j, b_j]$) and the current interval (dynamic interval, $[\mathrm{MIN}_j^p, \mathrm{MAX}_j^p]$) of the variables are used to generate new random points.

After this modification (using additional random numbers instead of opposite numbers), the random version of ODE (called RDE) is introduced. Now, we are ready to apply this algorithm to solve our test problems. All control parameters are kept the same to have a fair comparison. Results for the current algorithm are presented in Table 5.3 ($f_1 - f_{29}$) and Table 5.4 ($f_{30} - f_{58}$); also the results of DE and ODE (from Table 5.1) are repeated in these tables to ease the comparison among these three competitors (DE, ODE, and RDE). Two acceleration rates are reported in the last row of the Table 5.4, $\mathrm{AR_{ODE}} = \frac{\mathrm{NFC_{DE}}}{\mathrm{NFC_{ODE}}}$ and $\mathrm{AR_{RDE}} = \frac{\mathrm{NFC_{DE}}}{\mathrm{NFC_{RDE}}}$ compare DE with ODE and RDE, respectively.

As seen, ODE outperforms DE and RDE on 40 functions. DE performs better than ODE and RDE on 15 functions. The RDE can outperform DE (but not ODE) on just 3 functions f_{17}, f_{22}, and f_{35} (emphasized in boldface under the $\mathrm{AR_{RDE}}$ column). The average acceleration rate (DE vs. RDE) is 0.87; which means the RDE is 13% slower than its parent algorithm. The average success rate is almost the same for all of them (0.86, 0.86, and 0.87 for DE, ODE, and RDE, respectively).

Results analysis - Just by replacing the opposite numbers with additional random numbers - while the random numbers are generated uniformly in the variables dynamic intervals and the rest of the proposed algorithm is kept untouched - the average acceleration rate has dropped from 1.44 ($\mathrm{AR_{ODE}}$) to 0.87 ($\mathrm{AR_{RDE}}$) which is a 57% reduction in speed. This clearly demonstrates that the achieved improvements are due to usage of opposite points, and that the same level of improvement cannot be achieved via additional random sampling. We had already demonstrated both mathematically and experimentally why opposite numbers are more beneficial than pure random numbers (see chapter 3).

5.5.4 Experiment series 4: effect of population size

In order to investigate the effect of the population size, the same experiments (conducted in section 5.5.1 for $N_p = 100$) are repeated for $N'_p = N_p/2$ and $N'_p = 2N_p$. The results for

Table 5.3: Comparison of DE, ODE, and RDE. The best result is highlighted in **boldface** ($f_1 - f_{29}$).

		DE		*ODE*			*RDE*		
F	*D*	*NFC*	*SR*	*NFC*	*SR*	AR_{ODE}	*NFC*	*SR*	AR_{RDE}
f_1	30	87748	1	**47716**	1	1.83	115096	1	0.76
f_2	30	96488	1	**53304**	1	1.81	126780	1	0.76
f_3	20	177880	1	**168680**	1	1.05	231152	1	0.77
f_4	30	**403112**	1	–	0	–	506596	1	0.80
f_5	10	328844	**1**	**70389**	0.76	4.67	501875	0.96	0.66
f_6	30	113428	1	**69342**	0.96	1.64	149744	1	0.76
f_7	30	25140	1	**8328**	1	3.01	29096	1	0.86
f_8	30	169152	1	**98296**	1	1.72	222784	1	0.76
f_9	2	**4324**	1	4776	1	0.90	4904	1	0.88
f_{10}	4	**16600**	1	19144	1	0.87	21712	1	0.76
f_{11}	2	8016	1	**6608**	1	1.21	8248	1	0.97
f_{12}	3	**3376**	1	3580	1	0.94	3652	1	0.92
f_{13}	6	–	0	–	0	–	–	0	–
f_{14}	2	5352	1	**4468**	1	1.20	6292	1	0.85
f_{15}	30	101460	1	**70408**	1	1.44	138308	1	0.73
f_{16}	100	3608	1	**3288**	1	1.09	3924	1	0.92
f_{17}	4	549850	0.04	**311800**	0.12	1.76	381840	**0.16**	**1.44**
f_{18}	10	**191340**	**0.76**	213330	0.56	0.90	306900	0.60	0.62
f_{19}	30	385192	1	**369104**	1	1.04	498200	1	0.77
f_{20}	2	**4884**	1	5748	1	0.85	6192	1	0.79
f_{21}	30	187300	1	**155636**	1	1.20	244396	1	0.76
f_{22}	30	570290	0.28	**72250**	0.88	7.89	285108	**1**	**2.00**
f_{23}	30	41588	1	**23124**	1	1.80	54316	1	0.77
f_{24}	30	818425	0.12	**470359**	**1**	1.78	–	0	–
f_{25}	4	**13925**	0.80	15280	0.60	0.91	17038	**0.96**	0.82
f_{26}	4	–	0	–	0	–	–	0	–
f_{27}	4	–	0	–	0	–	–	0	–
f_{28}	4	4576	1	**4500**	1	1.02	5948	1	0.77
f_{29}	2	**10788**	1	11148	1	0.97	11588	1	0.93

$N_p = 50$ and $N_p = 200$ are given in Tables 5.5 and 5.6, respectively. In order to discuss the population size, the overall results of three tables (Table 5.1, Table 5.5, and Table 5.6) are summarized in Table 5.7.

For $N_p = 50$, the average success rate for DE and ODE is 0.79 and 0.77, respectively (DE performs marginally better than ODE). But, DE fails to solve 9 functions while ODE fails on 7. ODE outperforms DE on 35 functions; this number is 15 for DE. The average acceleration rate is 1.05 for this case (AR=60.16 for f_{33} is excluded as an exceptional case in order to avoid non-representative statistics). By carefully looking at the results, we can

Table 5.4: Continued from Table 5.3 ($f_{30} - f_{58}$).

		DE		*ODE*			*RDE*		
F	D	*NFC*	*SR*	*NFC*	*SR*	AR_{ODE}	*NFC*	*SR*	AR_{RDE}
f_{30}	2	1016	1	**996**	1	1.02	1084	1	0.94
f_{31}	30	411164	1	**337532**	1	1.22	927230	0.24	0.44
f_{32}	2	7976	1	**5092**	1	1.56	8332	1	0.96
f_{33}	5	2163	0.88	**2024**	**1**	1.07	2904	0.96	0.74
f_{34}	5	38532	1	**16340**	1	2.36	46080	1	0.84
f_{35}	2	2052	1	**1856**	1	1.11	2016	1	**1.02**
f_{36}	2	8412	1	**5772**	1	1.46	10860	1	0.77
f_{37}	2	5284	1	**4728**	1	1.12	5448	1	0.97
f_{38}	2	5280	1	**4804**	1	1.10	5540	1	0.95
f_{39}	2	3780	1	**3396**	1	1.11	3820	1	0.98
f_{40}	2	2424	1	**2152**	1	1.13	2400	1	1.00
f_{41}	10	19528	1	**15704**	1	1.24	23156	1	0.84
f_{42}	2	4780	1	**4684**	1	1.02	4924	1	0.97
f_{43}	3	**6852**	1	8484	1	0.81	8488	1	0.81
f_{44}	3	7036	1	**6172**	1	1.14	7396	1	0.95
f_{45}	2	3256	1	**3120**	1	1.04	3308	1	0.98
f_{46}	3	6184	1	**5472**	1	1.13	6676	1	0.93
f_{47}	2	2976	1	**2872**	1	1.03	3200	1	0.93
f_{48}	4	**1108**	1	1232	1	0.90	1388	1	0.80
f_{49}	2	**5232**	**1**	5956	0.92	0.88	5768	1	0.91
f_{50}	4	379900	**1**	**250260**	0.20	1.52	528632	1	0.72
f_{51}	10	–	0	**16681**	**0.84**	–	–	0	–
f_{52}	10	**14968**	1	15104	1	0.99	18328	1	0.82
f_{53}	2	7888	1	**4272**	1	1.85	9148	1	0.86
f_{54}	4	8856	1	**7504**	1	1.18	10632	1	0.83
f_{55}	9	**78567**	0.24	97536	0.56	0.81	103330	0.56	0.76
f_{56}	10	37824	1	**24260**	1	1.56	46800	1	0.80
f_{57}	2	**30704**	1	55980	0.76	0.55	37696	1	0.81
f_{58}	4	**16600**	1	19144	1	0.87	21344	1	0.78
Ave.			*0.86*		*0.86*	**1.44**		**0.87**	0.87

recognize that when the population size is reduced from 100 to 50, four functions (namely, f_3, f_5, f_{19}, and f_{31}) for which ODE has outperformed DE, are now (for population size 50) solved faster by DE. However, this was predictable because the dimension of those functions are 20, 10, 30, and 30, respectively, and $N_p = 50$ is a small population size to solve these functions. Many authors have proposed 10D as a proper value for the population size [Liu and Lampinen 2005; Storn 1996]. On the other hand, as we know, ODE reduces the population diversity by its selection method. For small population size, we need to reduce the jumping rate in order to control the diversity reduction. Here, the jumping rate was

Table 5.5: Comparison of DE and ODE ($N_p = 50$).

		DE		*ODE*					*DE*		*ODE*		
F	D	*NFC*	*SR*	*NFC*	*SR*	*AR*	F	D	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_1	30	32548	1	**25293**	0.92	1.29	f_{30}	2	628	1	**522**	1	1.20
f_2	30	36126	1	**28889**	0.92	1.25	f_{31}	30	**85136**	1	94082	0.88	0.90
f_3	20	**81630**	1	88272	1	0.92	f_{32}	2	3806	1	**2830**	1	1.34
f_4	30	–	0	–	0	–	f_{33}	5	64009	0.60	**1064**	1	60.16
f_5	10	**94627**	**0.60**	170200	0.24	0.57	f_{34}	5	16400	0.96	**11282**	1	1.45
f_6	30	42543	0.80	**36426**	0.76	1.17	f_{35}	2	1046	1	**1036**	1	1.00
f_7	30	8984	1	**3600**	1	2.50	f_{36}	2	3920	1	**3074**	1	1.28
f_8	30	63180	1	**52930**	0.80	1.19	f_{37}	2	2664	1	**2480**	1	1.07
f_9	2	**2248**	1	2378	1	0.95	f_{38}	2	2786	1	**2444**	1	1.14
f_{10}	4	**8456**	**0.96**	54136	0.84	0.16	f_{39}	2	1956	1	**1776**	1	1.10
f_{11}	2	3924	1	**3378**	1	1.16	f_{40}	2	1275	1	**1116**	1	1.14
f_{12}	3	1856	1	**1796**	1	1.03	f_{41}	10	9246	1	**7554**	1	1.22
f_{13}	6	–	0	–	0	–	f_{42}	2	2408	1	**2370**	1	1.02
f_{14}	2	2588	1	**2366**	1	1.09	f_{43}	3	**3512**	1	4018	1	0.87
f_{15}	30	37065	0.80	**32513**	**0.84**	1.14	f_{44}	3	3534	1	**3180**	1	1.11
f_{16}	100	1826	1	1782	1	1.02	f_{45}	2	1734	1	**1654**	1	1.05
f_{17}	4	–	0	–	0	–	f_{46}	3	3162	1	**2916**	1	1.08
f_{18}	10	**217870**	0.20	219790	0.20	0.99	f_{47}	2	1618	1	**1528**	1	1.06
f_{19}	30	**162700**	1	185882	1	0.88	f_{48}	4	**564**	1	776	1	0.73
f_{20}	2	**2614**	1	2804	1	0.93	f_{49}	2	**2809**	0.84	3308	**0.88**	0.91
f_{21}	30	64960	1	**63734**	1	1.02	f_{50}	4	–	0	**364300**	**0.08**	–
f_{22}	30	–	0	–	0	–	f_{51}	10	–	0	**136290**	**0.28**	–
f_{23}	30	15688	1	**12658**	1	1.24	f_{52}	10	**7160**	1	7262	1	0.99
f_{24}	30	–	0	–	0	–	f_{53}	2	4230	1	**2580**	1	1.64
f_{25}	4	**8866**	**0.72**	77539	0.52	0.11	f_{54}	4	4462	1	**3998**	1	1.12
f_{26}	4	–	0	–	0	–	f_{55}	9	271620	**0.40**	**182956**	0.36	1.48
f_{27}	4	–	0	–	0	–	f_{56}	10	18874	1	**12714**	1	1.48
f_{28}	4	2472	0.96	**2316**	1	1.06	f_{57}	2	**14550**	**0.92**	69472	0.60	0.21
f_{29}	2	**10420**	1	10884	1	0.96	f_{58}	4	**8456**	**0.96**	54136	0.84	0.16
$\mathrm{SR}_{ave}(DE) = 0.79, \mathrm{SR}_{ave}(ODE) = 0.77, \mathrm{AR}_{ave} = 1.05$													

kept the same (= 0.3) for all population sizes.

As mentioned before, for $N_p = 100$, DE and ODE show an equal average success rate (SR= 0.86), and they fail to solve an equal number of functions ($n_{(SR=0)} = 4$). But, ODE outperforms DE on 40 functions, whereas DE outperforms ODE on 15 functions. The average acceleration rate is 1.44 for this case.

For $N_p = 200$, ODE outperforms DE on all mentioned measures (0.86, 6, and 39 vs. 0.82, 7, and 15, respectively); and the average acceleration rate is 1.86. As shown in the last two rows of Table 5.7, ODE performs better than DE in terms of all performance measures.

Table 5.6: Comparison of DE and ODE ($N_p = 200$).

		DE		*ODE*					*DE*		*ODE*		
F	D	*NFC*	*SR*	*NFC*	*SR*	*AR*	F	D	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_1	30	234912	1	**103968**	1	2.26	f_{30}	2	1808	1	**1792**	1	1.01
f_2	30	259096	1	**116360**	1	2.23	f_{31}	30	–	0	811075	**0.28**	–
f_3	20	488544	1	**412808**	1	1.18	f_{32}	2	15784	1	**9152**	1	1.72
f_4	30	**809424**	**1**	–	0	–	f_{33}	5	4192	1	**3832**	1	1.09
f_5	10	983900	0.04	**149800**	**1**	6.57	f_{34}	5	80864	1	**27568**	1	2.93
f_6	30	307880	**1**	**155800**	0.88	1.98	f_{35}	2	3824	1	**3728**	1	1.03
f_7	30	71864	1	**17144**	1	4.19	f_{36}	2	17320	1	**10784**	1	1.60
f_8	30	457848	1	**211768**	1	2.16	f_{37}	2	10272	1	**9032**	1	1.14
f_9	2	**8200**	1	9200	1	0.89	f_{38}	2	10336	1	**9024**	1	1.15
f_{10}	4	**34400**	1	37688	1	0.91	f_{39}	2	7104	1	**6512**	1	1.09
f_{11}	2	15800	1	**12496**	1	1.26	f_{40}	2	4320	1	**3488**	1	1.24
f_{12}	3	**6632**	1	7104	1	0.93	f_{41}	10	39976	1	**31448**	1	1.27
f_{13}	6	–	0	–	0	–	f_{42}	2	8920	1	**8712**	1	1.02
f_{14}	2	10264	1	**7648**	1	1.34	f_{43}	3	**13664**	1	16384	1	0.83
f_{15}	30	280280	1	**173000**	1	1.62	f_{44}	3	13256	1	**12080**	1	1.10
f_{16}	100	6832	1	**6304**	1	1.08	f_{45}	2	6376	1	**6056**	1	1.05
f_{17}	4	**513930**	**0.08**	–	0	–	f_{46}	3	11776	1	**10656**	1	1.11
f_{18}	10	498090	0.84	**448140**	**0.88**	1.11	f_{47}	2	5872	1	**5424**	1	1.08
f_{19}	30	–	0	**927440**	**0.84**	–	f_{48}	4	**1904**	1	2424	1	0.79
f_{20}	2	**9824**	1	10368	1	0.95	f_{49}	2	**10184**	1	11840	1	0.86
f_{21}	30	526280	1	**368048**	1	1.43	f_{50}	4	**747848**	**1**	788810	0.88	0.95
f_{22}	30	987800	0.04	**144368**	**1**	6.84	f_{51}	10	–	0	**27220**	**1**	–
f_{23}	30	111792	1	**48304**	1	2.31	f_{52}	10	**30304**	1	31056	1	0.98
f_{24}	30	–	0	–	0	–	f_{53}	2	15216	1	**8392**	1	1.81
f_{25}	4	**25896**	0.92	30757	0.92	0.84	f_{54}	4	17088	1	**14760**	1	1.16
f_{26}	4	–	0	–	0	–	f_{55}	9	**174620**	0.36	272170	**0.44**	0.64
f_{27}	4	–	0	–	0	–	f_{56}	10	71760	1	**49088**	1	1.46
f_{28}	4	8856	1	**8512**	1	1.04	f_{57}	2	49816	1	**6896**	1	7.22
f_{29}	2	**20768**	1	22008	1	0.94	f_{58}	4	**33720**	1	37240	1	0.91
$\mathrm{SR}_{ave}(DE) = 0.82, \mathrm{SR}_{ave}(ODE) = 0.86, \mathrm{AR}_{ave} = 1.68$													

Results analysis - According to the results of section 5.5.1 and 5.5.2, for the majority of functions, ODE performs better when the dimension of the scalable problems increases. On the other hand, for higher dimensional problems a larger population size should be employed (e.g. $N_p = 10D$). According to results of this section, ODE performs better for larger population sizes.

Table 5.7: The summarized results from Table 5.1, Table 5.5, and Table 5.6. n_{DE} and n_{ODE} are the number of functions for which DE outperforms ODE and vice versa. $n_{SR=0}$ is the number of unsolved functions by the algorithm (SR= 0).

	DE			*ODE*			
N_p	SR_{ave}	$n_{SR=0}$	n_{DE}	SR_{ave}	$n_{SR=0}$	n_{ODE}	AR_{ave}
50	**0.79**	9	16	0.77	**7**	**35**	**1.05**
100	0.86	4	15	0.86	4	**40**	**1.44**
200	0.82	7	15	**0.86**	**6**	**39**	**1.86**
$\sum$		20	46		**17**	**114**	
Ave.	0.82			**0.83**			**1.45**

5.5.5 Experiment series 5: effect of various mutation operators

More than ten mutation strategies have been developed for DE [Storn and Price 1997a; Price et al. 2005; Feoktistov 2006]. Although, many researchers report results for one of these mutation strategies, most works [Storn and Price 1997a; Brest et al. 2006; Sun et al. 2005] use the standard one, namely DE/rand/1/bin, as we did. In this study, three other well-known mutation strategies, namely, DE/rand/1/exp, DE/rand/2/exp, and DE/rand/2/bin are selected to investigate the effect of the mutation operator. The results are presented in Table 5.8 (for $f_1 - f_{29}$) and Table 5.9 (for $f_{30} - f_{58}$). The overall results of these two tables and Table 5.1 (for DE/rand/1/bin) are summarized in compact form in Table 5.10 to ease the comparison.

For all mutation strategies, ODE performs better than DE by looking at the total number of function calls, average success rate, number of solved functions, number of functions for which ODE outperforms DE, and the average acceleration rate.

Results analysis - According to the Table 5.10, the best mutation strategy for DE and also for ODE is the DE/rand/1/bin. This confirms choosing mutation strategy DE/rand/1/bin as a standard operator by other researchers [Storn and Price 1997a; Brest et al. 2006; Sun et al. 2005]. Furthermore, it is noticeable that the average ac-

Table 5.8: Comparison of DE and ODE on three different mutation strategies ($f_1 - f_{29}$). The best result of each mutation strategy is emphasized in **boldface**.

		DE/rand/1/exp					**DE/rand/2/exp**					**DE/rand/2/bin**				
		DE		*ODE*			*DE*		*ODE*			*DE*		*ODE*		
F	D	*NFC*	*SR*	*NFC*	*SR*	*AR*	*NFC*	*SR*	*NFC*	*SR*	*AR*	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_1	30	86096	1	**47544**	1	1.81	675148	1	**74652**	1	9.05	683932	1	**76920**	1	8.89
f_2	30	95840	1	**53612**	1	1.88	742468	1	**84728**	1	8.76	741908	1	**84904**	1	8.74
f_3	20	**146336**	1	147652	1	0.99	781680	1	**393652**	1	1.99	–	0	**481992**	**1**	–
f_4	30	**302072**	**1**	–	0	–	–	0	**343340**	**1**	–	–	0	**463880**	**1**	–
f_5	10	391528	**1**	**70983**	0.92	5.52	–	0	**186370**	**0.72**	–	–	0	**172860**	**0.72**	–
f_6	30	112892	**1**	**69475**	0.96	1.62	923540	0.16	**126370**	**0.92**	7.30	29968	1	29760	1	1.00
f_7	30	25520	1	**8032**	1	3.18	296028	1	**11316**	1	26.16	302972	1	**11736**	1	25.81
f_8	30	168468	1	**98392**	1	1.71	–	0	**159172**	**1**	–	–	0	**159296**	**1**	–
f_9	2	**4268**	1	4684	1	0.91	**5512**	1	5712	1	0.97	**5676**	1	5776	1	0.98
f_{10}	4	**17020**	1	19216	1	0.89	30292	1	**29896**	1	1.01	–	0	–	0	–
f_{11}	2	8052	1	**6464**	1	1.25	12828	1	**7744**	1	1.66	12916	1	**7404**	1	1.74
f_{12}	3	**3552**	1	3612	1	0.98	4352	1	**4146**	1	1.05	44444	1	**4196**	1	1.06
f_{13}	6	–	0	–	0	–	–	0	–	0	–	–	0	–	0	–
f_{14}	2	5236	1	**4260**	1	1.23	7928	1	**5004**	1	1.58	8020	1	**4920**	1	1.63
f_{15}	30	100264	1	**71040**	1	1.41	948920	0.96	**188496**	**1**	5.03	963740	0.84	**195096**	**1**	4.94
f_{16}	100	3496	1	**3148**	1	1.11	4440	1	**3616**	1	1.23	4588	1	**3724**	1	1.23
f_{17}	4	**255230**	**0.20**	–	0	–	–	0	–	0	–	–	0	–	0	–
f_{18}	10	**189830**	**0.60**	279260	0.32	0.68	–	0	**825330**	**0.08**	–	–	0	–	0	–
f_{19}	30	331864	1	**331040**	1	1.00	–	0	–	0	–	–	0	–	0	–
f_{20}	2	**5196**	1	6032	1	0.86	**9628**	1	10052	1	0.96	**9612**	1	9704	1	0.99
f_{21}	30	**185176**	1	155572	1	1.19	–	0	**281560**	**1**	–	–	0	**280328**	**1**	–
f_{22}	30	560100	0.08	**71296**	**0.96**	7.86	–	0	**97824**	**1**	–	–	0	**98380**	**1**	–
f_{23}	30	41112	1	**23024**	1	1.79	366972	1	**39144**	1	9.37	366852	1	**38552**	1	9.52
f_{24}	30	–	0	–	0	–	–	0	–	0	–	–	0	–	0	–
f_{25}	4	**14625**	**0.96**	85507	0.52	0.17	**18921**	**0.76**	21830	0.40	0.87	**18421**	**0.76**	104890	0.44	0.18
f_{26}	4	–	0	–	0	–	–	0	–	0	–	–	0	–	0	–
f_{27}	4	–	0	–	0	–	–	0	–	0	–	–	0	–	0	–
f_{28}	4	4800	1	**4368**	1	1.10	7944	1	**6800**	1	1.17	8032	1	**6832**	1	1.18
f_{29}	2	**10724**	1	10836	1	0.99	15280	1	**14644**	1	1.04	15168	1	**14884**	1	1.02

Table 5.9: Continued from Table 5.8 ($f_{30} - f_{58}$).

		DE/rand/1/exp					**DE/rand/2/exp**					**DE/rand/2/bin**				
		DE		*ODE*			*DE*		*ODE*			*DE*		*ODE*		
F	D	*NFC*	*SR*	*NFC*	*SR*	*AR*	*NFC*	*SR*	*NFC*	*SR*	*AR*	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_{30}	2	1072	1	**980**	1	1.10	1240	1	**1108**	1	1.12	1292	1	**1120**	1	1.15
f_{31}	30	365492	**1**	**287330**	0.92	1.27	–	0	**653600**	**0.40**	–	–	0	**613900**	**0.24**	–
f_{32}	2	8004	1	**4856**	1	1.65	15956	1	**6100**	1	2.62	14848	1	**5776**	1	2.57
f_{33}	5	2204	0.96	**2068**	**1**	1.07	2136	1	**1944**	1	1.09	2108	1	**1896**	1	1.11
f_{34}	5	35888	1	**18836**	1	1.90	131756	1	**22968**	1	5.74	145348	1	**23196**	1	6.26
f_{35}	2	2052	1	**1952**	1	1.05	2528	1	**2236**	1	1.13	2492	1	**2144**	1	1.16
f_{36}	2	8320	1	**5660**	1	1.47	15904	1	**6680**	1	2.38	16144	1	6408	1	2.52
f_{37}	2	5320	1	**4668**	1	1.14	6756	1	**5320**	1	1.27	6588	1	**5252**	1	1.25
f_{38}	2	5260	1	**4676**	1	1.12	6636	1	**5304**	1	1.25	6844	1	**5344**	1	1.28
f_{39}	2	3736	1	**3288**	1	1.14	4620	1	**3676**	1	1.26	4784	1	**3780**	1	1.27
f_{40}	2	2388	1	**1984**	1	1.20	2960	1	**2292**	1	1.29	3288	1	**2276**	1	1.44
f_{41}	10	19452	1	**15452**	1	1.26	41212	1	**22908**	1	1.80	41440	1	**23100**	1	1.80
f_{42}	2	**4564**	1	4568	1	0.99	6000	1	**5384**	1	1.11	5988	1	**5360**	1	1.12
f_{43}	3	**6892**	1	7868	1	0.88	**9864**	1	11588	1	0.85	**9592**	1	11480	1	0.84
f_{44}	3	7092	1	**6196**	1	1.14	9160	1	**7208**	1	1.27	9096	1	**7220**	1	1.26
f_{45}	2	3292	1	**3168**	1	1.04	4072	1	**3568**	1	1.14	4064	1	**3568**	1	1.14
f_{46}	3	6100	1	**5536**	1	1.10	8304	1	**6748**	1	1.23	8156	1	**6660**	1	1.22
f_{47}	2	3016	1	**2924**	1	1.03	3748	1	**3244**	1	1.16	3708	1	**3144**	1	1.18
f_{48}	4	**1032**	1	1348	1	0.76	**892**	1	1244	1	0.72	**1100**	1	1132	1	0.97
f_{49}	2	**5343**	0.92	5988	**1**	0.89	**7344**	1	8504	1	0.86	**7525**	0.96	8496	**1**	0.86
f_{50}	4	450960	**1**	**359540**	0.16	1.25	–	0	–	0	–	–	0	–	0	–
f_{51}	10	–	0	**17613**	**0.92**	–	–	0	**23385**	**0.80**	–	–	0	**56289**	**0.74**	–
f_{52}	10	**15028**	1	15140	1	0.99	30468	1	**24504**	1	1.24	30496	1	**24440**	1	1.25
f_{53}	2	7652	1	**4384**	1	1.75	13280	1	**5192**	1	2.56	12428	1	**5276**	1	2.36
f_{54}	4	8756	1	**7624**	1	1.15	12520	1	**9196**	1	1.36	12664	1	**9316**	1	1.36
f_{55}	9	**85360**	0.20	163280	**0.52**	0.52	**301510**	0.48	324204	**0.68**	0.93	299239	0.32	**279310**	**0.76**	1.07
f_{56}	10	37860	1	**24972**	1	1.52	93652	1	**36384**	1	2.57	90764	1	**38016**	1	2.39
f_{57}	2	33428	1	**6178**	**0.92**	5.41	62468	**1**	**45824**	0.96	1.36	84120	1	**5404**	1	15.56
f_{58}	4	**16888**	1	18352	1	0.92	30292	1	**29896**	1	1.01	**29824**	1	31168	1	0.96

Table 5.10: The summarized results from Tables 5.1, 5.8, and 5.9. $\sum NFC_i$ is the total of number of function calls (just for the functions which all 8 competitors could solve). $n_{SR=0}$ is the number of unsolved functions (SR = 0). n_{DE} and n_{ODE} are the number of functions for which DE outperforms ODE and vice versa. N is the number of functions for which the algorithm could outperform seven other algorithms. AR_{ave} is the average acceleration rate.

	DE					*ODE*					
Mutation Strategy	$\sum NFC_i$	SR_{ave}	$n_{SR=0}$	n_{DE}	N	$\sum NFC_i$	SR_{ave}	$n_{SR=0}$	n_{ODE}	N	AR_{ave}
$DE/rand/1/bin$	853,979	0.86	4	15	7	**656,274**	0.86	4	**40**	**14**	1.42
$DE/rand/1/exp$	857,672	**0.86**	**5**	17	8	**743,584**	0.85	6	**37**	**16**	1.51
$DE/rand/2/exp$	4,865,187	0.71	15	7	0	**1,207,478**	**0.83**	**7**	**44**	**7**	2.78
$DE/rand/2/bin$	4,070,189	0.69	17	8	1	**1,119,580**	**0.81**	**9**	**41**	**2**	3.03

celeration rate is even higher for other mutation strategies. For the mutation strategy $DE/rand/2/bin$, the average acceleration rate 3.03 is the highest.

5.5.6 Experiment series 6: proper setting of jumping rate J_r

In the proposed algorithm, a new control parameter, J_r, is added to DE parameters (F,C_r, and N_p). Although this parameter was fixed for all experiments, the performance of ODE can vary for different J_r values. The jumping rate for our current study was set to $J_r = 0.3$ without any effort to find an optimal value. In some trials, we observed that a jumping rate higher than 0.6 is not suitable for many functions and causes a premature convergence or an unusual growing of the number of function evaluations. On the other hand, $J_r = 0$ means no usage of opposition. With this method, simply the mean value of the 0 and 0.6, ($J_r = 0.3$), was selected for all conducted experiments as a default value.

In this section, we try to find an optimal jumping rate (J_{rop}) for each test function from a discrete set of jumping rate values to answer the question whether a general recommendation for J_r setting can be offered. Now, we are faced with this fundamental question: In order to find the optimal jumping rate, should we look for the minimum number of function

calls (NFC) or the maximum success rate (SR)? Both measures are important factors in an optimization process. So, two individual objectives should be considered simultaneously. In order to combine these two measures, a new measure, called success performance (SP), has been introduced as follows [Suganthan et al. 2005]:

$$\text{SP} = \frac{\text{mean (NFC for successful runs)}}{\text{SR}}. \tag{5.5}$$

By this definition, the two following algorithms have equal performances (SP=100):

Algorithm A: mean (NFC for successful runs)=50 and SR=0.5,

Algorithm B: mean (NFC for successful runs)=100 and SR=1.

SP gives an equal importance weight to NFC and SR. But depending on different applications each of them can be more important than other one. Sometimes success rate is more crucial than convergence speed and vice versa. For our experiments, gathering results for unsuccessful case is more time consuming because the algorithm should meet a maximum number of function calls for termination.

Now, we repeat the conducted experiments in section 5.5.1 for $J_r \in (0, 0.6]$ with a step size of 0.1 (i.e. 50 trials per function per jumping rate value $J_r \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$). Due to space limitations, we do not show all results but just the obtained optimal value for the jumping rate with respect to the success performance as given in Table 5.11.

As seen, the optimal values for the jumping rate are distributed over the discrete interval $(0, 0.6]$. However, jumping rates of 0.3 and 0.6 occur more frequently than other values in this table. Higher jumping rates mostly belong to the low dimensional functions and lower ones to the high dimensional functions (but this is probably not a general phenomenon). The average value of the obtained optimal jumping rates ($\bar{J}_{rop}$) is equal to 0.37 for our test functions.

Some sample graphs (SP vs. J_r) are shown in Figure 5.4 to illustrate the effect of jumping rate on success performance. The point specified by $J_r = 0$ indicates the success

Table 5.11: Optimal jumping rate J_{rop} for all test functions with respect to the success performance (SP) on interval $(0, 0.6]$ with step-size of 0.1.

F	D	J_{rop}	F	D	J_{rop}	F	D	J_{rop}	F	D	J_{rop}
f_1	30	0.3	f_{16}	100	0.5	f_{31}	30	0.1	f_{46}	3	0.6
f_2	30	0.3	f_{17}	4	0.3	f_{32}	2	0.4	f_{47}	2	0.6
f_3	20	0.3	f_{18}	10	0.2	f_{33}	5	0.6	f_{48}	4	0.6
f_4	30	0.1	f_{19}	30	0.2	f_{34}	5	0.6	f_{49}	2	0.1
f_5	10	0.2	f_{20}	2	0.1	f_{35}	2	0.5	f_{50}	4	0.1
f_6	30	0.3	f_{21}	30	0.5	f_{36}	2	0.6	f_{51}	10	0.2
f_7	30	0.6	f_{22}	30	0.2	f_{37}	2	0.6	f_{52}	10	0.3
f_8	30	0.3	f_{23}	30	0.3	f_{38}	2	0.6	f_{53}	2	0.6
f_9	2	0.1	f_{24}	30	0.3	f_{39}	2	0.5	f_{54}	4	0.6
f_{10}	4	0.2	f_{25}	4	0.3	f_{40}	2	0.5	f_{55}	9	0.6
f_{11}	2	0.4	f_{26}	4	–	f_{41}	10	0.6	f_{56}	10	0.6
f_{12}	3	0.1	f_{27}	4	–	f_{42}	2	0.4	f_{57}	2	0.5
f_{13}	6	–	f_{28}	4	0.3	f_{43}	3	0.1	f_{58}	4	0.2
f_{14}	2	0.5	f_{29}	2	0.1	f_{44}	3	0.5			
f_{15}	30	0.3	f_{30}	2	0.6	f_{45}	2	0.4			

performance of the DE; the rest of points $(0.1, 0.2, 0.3, 0.4, 0.5, 0.6)$ show the success performance of ODE. As mentioned before, we can observe a sharp increase in the SP for hard functions (e.g. f_5, f_8, f_{22}, f_{31}, and f_{51}) using higher jumping rates. Also, the SP decreases for easy functions by increasing the jumping rate (see f_7, f_{41}, and f_{56}). Almost a smooth behavior for all functions is recognizable for $J_r \in \{0.1, 0.2, 0.3, 0.4\}$ (it was observed even for many functions whose graphs are not presented here). Hence, working in this interval ($[0.1, 0.4]$) could be reasonable for unknown optimization problems.

Results analysis - Like DE's control parameters, optimal jumping rate should have a problem-oriented value. The conducted experiments suggest the range of $[0.1, 0.4]$ for an unknown optimization problem. A first attempt can be conducted with $J_r = 0.37$ ($\bar{J}_{rop}$). Furthermore, for high dimensional problems, a smaller jumping rate can be suggested.

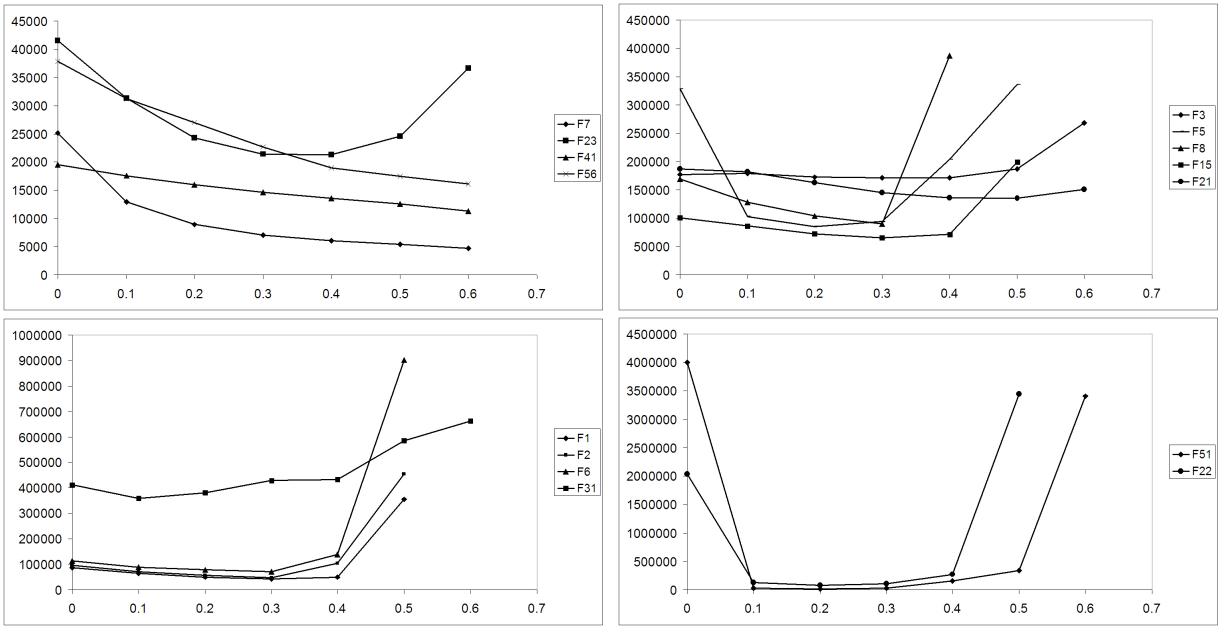


Fig. 5.4: Graphs of success performance (SP) vs. jumping rate ($J_r \in (0, 0.6]$ with step size of 0.1) for sample functions. $J_r = 0$ shows the SP of the DE; the rest of points $(0.1, 0.2, 0.3, 0.4, 0.5, 0.6)$ show the SP of the ODE.

5.5.7 Experiment series 7: comparison with DE and FADE

The primary purpose here is to introduce the notion of the opposition into the design and implementation of the population-based algorithms, differential evolution in particular, and demonstrate its benefits. Many other extensions of DE, if not all, can also be reconsidered to incorporate the opposition concept. In this sense, ODE should be regarded as an example and not as a competitor to other DE versions or other optimizers. However, in the current section and the following two sections, in order to assess ODE's performance, it is compared to the parent algorithm (DE) and also to five other well-known evolutionary optimizers in term of solution quality.

ODE is compared with the fuzzy adaptive differential evolution (FADE) method of Liu and Lampinen [2005] in this section. FADE employs a fuzzy logic controller to set the

mutation and crossover rates. Liu and Lampinen tested FADE for 10 well-known benchmark functions, of which we have 9 in our test set. The comparison strategy is different for this experiment. The algorithms are run 100 times. Subsequently, for an equal (fixed) number of function calls the average and standard deviation of the best solutions are calculated for the purpose of comparison. The same settings for parameters [Liu and Lampinen 2005] have been used in the current experiment to assure a fair comparison. The population size is equal to 10D and instead of using the generation number for DE and ODE an equal number of function evaluations has been used ($N_p \times \#Gen.$) as the termination criteria (since in each generation the numbers of function calls for DE and ODE are not equal). The dimension of the functions, corresponding generation numbers, and obtained results (best mean and standard deviation of 100 runs) for DE, ODE, and FADE are given in Table 5.12. Results for FADE are taken from [Liu and Lampinen 2005, Table 6, p. 459]. A two-tailed t-test at a 0.05 significance level has been used to compare results of ODE against those of DE and FADE.

Results analysis - According to the t-test, ODE performs better than DE on 9 functions (out of 16). There is no difference for the rest of functions. ODE surpasses FADE on 12 functions, and they perform the same on the rest. So, DE and FADE cannot show better results than ODE even for one function. Although, the comparison of a non-adaptive algorithm (ODE) to an adaptive one (FADE) is not fair but interestingly the results confirm that ODE outstandingly performs not only better than DE but also better than FADE.

5.5.8 Experiment series 8: comparison with Adaptive LEP and Best Lévy

Lee and Yao [2004] proposed an evolutionary programming (EP) algorithm using adaptive as well as nonadaptive Lévy mutations. They called the approaches Adaptive LEP and

Table 5.12: Comparison of DE, ODE, and Fuzzy Adaptive DE (FADE). Mean best and standard deviation (Std Dev) of 100 runs are reported. For DE and ODE, equal number of function calls are used instead of generation numbers ($N_p \times \#Gen.$). Two-tailed t-test is used to compare ODE against DE and FADE. '†' indicates that the t value of 99 degree of freedom is significant at a 0.05 level of significance. f_{min} indicates optimal minimum of the function.

				DE	**ODE**	**FADE**
F	f_{min}	D	$\#\ Gen.$	Mean Best (Std Dev)	Mean Best (Std Dev)	Mean Best (Std Dev)
f_1	0	3	50	3.49×10^{-7} $(5.12 \times 10^{-7})^{\dagger}$	4.79×10^{-8} (5.81×10^{-8})	1.18×10^{-5} $(1.88 \times 10^{-10})^{\dagger}$
		50	5000	1.25×10^{-13} $(4.88 \times 10^{-14})^{\dagger}$	1.73×10^{-64} (1.59×10^{-64})	2.35×10^{-10} $(2.97 \times 10^{-21})^{\dagger}$
f_4	0	2	50	8.0×10^{-3} $(2.93 \times \times 10^{-2})$	$4.8 \times \times 10^{-3}$ $(1.80 \times \times 10^{-2})$	2.2×10^{-3} (7.73×10^{-5})
		50	7000	5.30 $(9.56 \times 10^{-1})^{\dagger}$	5.01×10^{-1} (8.90×10^{-1})	$4.16 \times 10^{+1}$ $(1.82 \times 10^{-2})^{\dagger}$
f_5	0	2	3000	0 (0)	0 (0)	0 (0)
		50	10000	$3.43 \times 10^{+2}$ $(0.98 \times 10^{+1})^{\dagger}$	$6.56 \times 10^{+1}$ $(6.80 \times 10^{+1})$	$2.58 \times 10^{+2}$ $(9.17 \times 10^{+1})^{\dagger}$
f_6	0	2	180	5.19×10^{-4} (1.9×10^{-3})	1.49×10^{-4} (1.0×10^{-3})	2.4×10^{-3} $(1.20 \times 10^{-5})^{\dagger}$
		50	5000	6.13×10^{-11} $(2.02 \times 10^{-11})^{\dagger}$	1.53×10^{-62} (2.38×10^{-62})	5.78×10^{-1} $(3.5 \times 10^{-3})^{\dagger}$
f_8	0	3	50	1.07×10^{-2} $(7.4 \times 10^{-3})^{\dagger}$	4.8×10^{-3} (3.7×10^{-3})	1.56×10^{-1} $(1.20 \times 10^{-2})^{\dagger}$
		50	5000	2.33×10^{-6} $(2.33 \times 10^{-6})^{\dagger}$	7.02×10^{-15} (1.66×10^{-15})	5.9×10^{-2} $(1.23 \times 10^{-6})^{\dagger}$
f_{11}	-1	2	80	-1 (0)	-1 (0)	-0.9896 $(1.3 \times 10^{-3})^{\dagger}$
f_{23}	0	5	30	0 (0)	0 (0)	0 (0)
		50	5000	0 (0)	0 (0)	0 (0)
f_{24}	0	30	5000	5.1×10^{-3} $(9.69 \times 10^{-4})^{\dagger}$	8.70×10^{-4} (3.30×10^{-4})	9.16 $(1.50 \times 10^{-1})^{\dagger}$
		50	5000	1.64×10^{-2} $(2.4 \times 10^{-3})^{\dagger}$	9.04×10^{-4} (3.41×10^{-4})	$1.90 \times 10^{+1}$ $(2.92 \times 10^{-1})^{\dagger}$
f_{42}	3	2	50	3 (0)	3 (0)	3.0001 $(3.35 \times 10^{-7})^{\dagger}$

Best Lévy, respectively. For the first time, a Lévy mutation instead of Gaussian mutation was employed in EP. According to the reported results, their algorithm performs better than classical EP in the case of functions with many local optima. These EPs new versions are tested for 14 benchmark functions, of which we have 12 in our test set. The comparison strategy is the same as previous experiment. The same settings for generation numbers, problem dimensions, and population size ($N_p = 100$) [Lee and Yao 2004] have been utilized. The dimension of the functions, corresponding generation numbers, and obtained results (best mean and standard deviation of 50 runs [1]) for ODE, Adaptive LEP and Best Lévy are given in Table 5.13. Results for Adaptive LEP and Best Lévy are taken from [Lee and Yao 2004, Table 3, p.12].

Results analysis - According to the t-test, ODE performs better than Adaptive LEP and Best Lévy on 7 and 8 functions (out of 12), respectively. While, Adaptive LEP and Best Lévy outperform ODE on 2 functions, individually. Over the rest of functions, the results are the same with respect to two-tailed t-test.

5.5.9 Experiment series 9: comparison with FEP and CEP

The fast evolutionary programming (FEP) was proposed in [Yao et al. 1999] and was compared to the classical evolutionary programming (CEP). A 23-function test suite was utilized to compare the algorithms. Our test suite contains 17 of them. The results to solve these 17 functions by ODE, FEP and CEP are presented in Table 5.14. In order to support a fair comparison, the same settings for the generation numbers, problem dimensions, and population size ($N_p = 100$) [Yao et al. 1999] have been utilized. Results for FEP and CEP are taken from [Yao et al. 1999, Table II-IV].

[1] The same setting as for Adaptive LEP and Best Lévy in [Lee and Yao 2004].

Table 5.13: Comparison of ODE, Adaptive LEP, and Best Lévy. Mean Best and standard deviation (Std Dev) of 50 runs are reported. For ODE, equal number of function calls are used instead of generation numbers ($N_p \times \#Gen.$). Two-tailed t-test is used to compare ODE against Adaptive LEP and Best Lévy. '†' indicates that the t value of 49 degree of freedom is significant at a 0.05 level of significance. '‡' stands for negative t, which means the corresponding algorithm performs better than ODE. f_{min} indicates optimal minimum of the function.

				ODE	Adaptive LEP	Best Lévy
F	f_{min}	D	$\#$ *Gen.*	Mean Best (Std Dev)	Mean Best (Std Dev)	Mean Best (Std Dev)
f_1	0	30	1500	4.48×10^{-27} (7.95×10^{-27})	6.32×10^{-4} $(7.6 \times 10^{-5})^{\dagger}$	6.59×10^{-4} $(6.4 \times 10^{-5})^{\dagger}$
f_3	0	30	1500	0.37 (0.35)	0.041 $(0.06)^{\ddagger}$	30.63 $(22.11)^{\dagger}$
f_4	0	30	1500	24.89 (1.20)	43.40 $(31.52)^{\dagger}$	57.75 $(41.60)^{\dagger}$
f_5	0	30	1500	66.94 (27.40)	5.85 $(2.07)^{\ddagger}$	12.50 $(2.29)^{\ddagger}$
f_6	0	30	1500	4.44×10^{-4} (1.8×10^{-3})	2.4×10^{-2} $(2.8 \times 10^{-2})^{\dagger}$	1.8×10^{-2} $(1.7 \times 10^{-2})^{\dagger}$
f_8	0	30	1500	4.42×10^{-14} (3.09×10^{-14})	1.9×10^{-2} $(1.0 \times 10^{-3})^{\dagger}$	3.1×10^{-2} $(2.0 \times 10^{-3})^{\dagger}$
f_{14}	-1.03162	2	30	-1.03 (2.35×10^{-5})	-1.031 (0.0)	-1.031 (0.0)
f_{26}	-10.1532	4	100	-10.1012 (1.68×10^{-1})	-9.54 $(1.69)^{\dagger}$	-9.95 $(0.99)^{\dagger}$
f_{27}	-10.4029	4	100	-10.2904 (7.17×10^{-1})	-10.30 (0.74)	-10.40 $(1.0 \times 10^{-4})^{\ddagger}$
f_{28}	-10.5364	4	100	-10.7003 (3.59×10^{-15})	-10.54 $(4.9 \times 10^{-5})^{\dagger}$	-10.54 $(3.1 \times 10^{-3})^{\dagger}$
f_{42}	3	2	30	3.0001 (7.77×10^{-5})	3.000 (0.000)	3.000 (0.000)
f_{46}	0	30	1500	7.15×10^{-23} (2.00×10^{-22})	6.0×10^{-6} $(1.0 \times 10^{-6})^{\dagger}$	3.0×10^{-5} $(4.0 \times 10^{-6})^{\dagger}$

Table 5.14: Comparison of ODE, FEP, and CEP. Mean Best and standard deviation (Std Dev) of 50 runs are reported. For ODE, equal number of function calls are used instead of generation numbers ($N_p \times \#Gen.$). Two-tailed t-test is used to compare ODE against FEP and CEP. '†' indicates that the t value of 49 degree of freedom is significant at a 0.05 level of significance. '‡' stands for negative t, which means the corresponding algorithm performs better than ODE. f_{min} indicates optimal minimum of the function.

				ODE	FEP	CEP
F	f_{min}	D	$\#$ *Gen.*	Mean Best (Std Dev)	Mean Best (Std Dev)	Mean Best (Std Dev)
f_1	0	30	1500	4.48×10^{-27} (7.95×10^{-27})	5.7×10^{-4} (1.3×10^{-4})†	2.20×10^{-4} (5.9×10^{-4})†
f_3	0	30	5000	8.85×10^{-11} (1.62×10^{-10})	1.6×10^{-2} (1.4×10^{-2})†	5.0×10^{-2} (6.6×10^{-2})†
f_4	0	30	20000	24.51 (1.11)	5.06 (5.87)‡	6.17 (13.61)‡
f_5	0	30	5000	11.61 (11.67)	4.6×10^{-2} (1.2×10^{-2})‡	89.0 (23.1)†
f_6	0	30	2000	8.86×10^{-4} (3.55×10^{-3})	1.6×10^{-2} (2.2×10^{-2})†	8.6×10^{-2} (0.12)†
f_8	0	30	1500	4.4×10^{-14} (3.09×10^{-14})	1.8×10^{-2} (2.1×10^{-3})†	9.2 (2.8)†
f_{14}	-1.03162	2	100	-1.03 (4.48×10^{-16})	-1.03 (4.9×10^{-7})	-1.03 (4.9×10^{-7})
f_{20}	0.398	2	100	3.97×10^{-1} (2.24×10^{-16})	3.98×10^{-1} (1.5×10^{-7})	3.98×10^{-1} (1.5×10^{-7})
f_{21}	0	30	2000	1.97×10^{-11} (1.29×10^{-11})	8.1×10^{-3} (7.7×10^{-4})†	2.6×10^{-3} (1.7×10^{-4})†
f_{22}	0	30	5000	8.23×10^{-2} (0.53)	0.3 (0.5)†	2.0 (1.2)†
f_{23}	0	30	1500	0 (0)	0 (0)	577.76 (1125.76)†
f_{24}	0	30	3000	1.4×10^{-3} (5.88×10^{-4})	7.6×10^{-3} (2.6×10^{-3})	1.8×10^{-3} (6.4×10^{-3})
f_{25}	0.000307	4	4000	1.95×10^{-4} (3.52×10^{-4})	5.0×10^{-4} (3.2×10^{-4})	4.7×10^{-4} (3.0×10^{-4})
f_{26}	-10.1532	4	100	-10.0008 (0.71)	-5.52 (1.59)†	-6.86 (2.67)†
f_{27}	-10.4029	4	100	-10.2904 (0.71)	-5.52 (2.12)†	-8.27 (2.95)†
f_{28}	-10.5364	4	100	-10.7003 (3.58×10^{-15})	-6.57 (3.14)†	-9.10 (2.92)†
f_{42}	3	2	100	3.0 (0)	3.02 (0.11)	3.0 (0)

Results analysis - As seen in Table 5.14, ODE outperforms FEP and CEP on 9 and 11 functions (out of 17) while FEP and CEP just on 2 and 1 functions present better results than ODE, respectively. Over the rest of functions, they perform almost the same.

5.6 Enhancement Directions

Further attempts are conducted to introduce the following two enhancement directions for ODE. The reader is refereed to Appendix B for comprehensive details.

First, instead of the opposite points, quasi-opposite points are suggested to be used. A quasi-opposite point is a randomly generated point between the center of search interval and the opposite point. This new extension is called quasi-oppositional DE (QODE) [Rahnamayan et al. 2007c]. The QODE employs exactly the same schemes of ODE for population initialization and generation jumping. A test suite with 15 benchmark functions has been employed to compare performance of DE, ODE, and QODE experimentally. QODE presents promising results.

Second, a time varying jumping rate (TVJR) model for ODE has been proposed [Rahnamayan et al. 2007a]. According to this model, the jumping rate changes during the evolution based on the number of function evaluations. Results confirm that a higher jumping rate is more desirable during the exploration than during the exploitation.

5.7 A Sample Application

In order to introduce a sample application of ODE, a new optimization-based image thresholding approach is proposed (see Chapter 6) [Rahnamayan et al. 2006a]. Then micro-DE and micro-ODE (DE and ODE with very small population size, $Np = 5$) have been compared over thresholding of 16 test images. The results confirm that the micro-DE is accelerated 13% just by employing the opposition-based population initialization. Furthermore,

the proposed thresholding approach is compared with the Kittler method.

5.8 Summary

The results of the conducted empirical studies in this chapter can be summarized as follows:

- DE and ODE were compared for different problem dimensions (D/2, D, and 2D); the results confirm that ODE performs better over high dimensional and scalable problems. For these kinds of problems a large population size is required. On the other hand, ODE performs better with larger population sizes. These two facts support each other and make ODE more suitable for higher dimensional problems. Further study is required to solidify this statement.

- By replacing opposite points with uniformly generated random points in the same variable range, the acceleration rate was reduced by 57%. Therefore, the contribution of opposite points to the acceleration process was confirmed and was not reproducible by additional random sampling.

- DE and ODE were compared for different population sizes ($N_p/2$, N_p, and $2N_p$). ODE performed better for larger population sizes, which is usually required for higher dimensional problems. In order to achieve better results for smaller population sizes, low jumping rates are suggested.

- Comparison of DE and ODE over various mutation strategies was also conducted. For all mutation strategies, ODE performed superior to DE with respect to the number of function calls, average success rate and other performance measures, such as the number of unsolved functions.

- The influence of the jumping rate was studied and the range $[0.1, 0.4]$ was recommended for an unknown optimization problem. Most of the functions presented a

reliable acceleration improvement and almost a smooth behavior in this interval. Although, the optimal jumping rate can be somewhere out of this range, higher jumping rates are generally not recommended.

- ODE was compared with five other optimizers (other than the classical DE) in terms of solution accuracy. Its competitors were Fuzzy Adaptive DE (FADE), Adaptive LEP, Best Lévy, Fast Evolutionary Programming (FEP), and Classical Evolutionary Programming (CEP). Overall, ODE accomplished better than its six competitors. Although, comparison of the non-adaptive algorithm (ODE) with the adaptive ones (FADE and LEP) may not be fair.

- By employing a comprehensive set of benchmark functions, the parent algorithm (DE) and ODE were compared on 462 optimization exercises during the seven experiment series. Overall, ODE presented promising results in terms of convergence speed and solution accuracy while the robustness was almost the same as DE's.

- By replacing opposite points with quasi-opposite points and utilizing lower jumping rates, the promising results were obtained (see Appendix B). Results of applying DE, ODE, and QODE to solve 30 test problems show that QODE outperforms DE and ODE on 22 functions. But still, further investigations are compulsory to have a solid conclusion about QODE's capabilities.

- As another enhancement direction for ODE, the time varying jumping rate was proposed and two behaviorally reverse versions (linearly decreasing and increasing functions) were compared with the constant setting (see Appendix B). The results confirm that the linearly decreasing jumping rate performs better than constant setting and also than linearly increasing policy.

Chapter 6

Image Thresholding Using micro-ODE

The first is when an opposite has been defined through its opposite, e.g. good through evil: for opposites are always simultaneous by nature. Some people think, also, that both are objects of the same science, so that the one is not even more intelligible than the other. One must, however, observe that it is perhaps not possible to define some things in any other way, e.g. the double without the half, and all the terms that are essentially relative: for in all such cases the essential being is the same as a certain relation to something, so that it is impossible to understand the one term without the other, and accordingly in the definition of the one the other too must be embraced. One ought to learn up all such points as these, and use them as occasion may seem to require.

– Aristotle (322 – 384 BC)

6.1 Introduction

In many image processing applications, the crucial role of the image thresholding can be observed (e.g., medical image processing [Rahnamayan et al. 2005a]). Numerous thresholding techniques have already been proposed [Sezgin and Sankur 2004; Cheng et al. 2001; Tizhoosh 2005a; Rahnamayan et al. 2006h]. However, almost all of them are application- or domain-oriented solutions, suffering from lack of universality. Therefore, this research field is still open to investigation and introduction of new robust and universal techniques.

In this chapter, a new thresholding technique is proposed which generates corresponding binary image by minimizing the dissimilarity between the input and the output images. Hence, the grey-level input image itself is directly used to measure the quality of the threshoded image; thus, this method can be introduced as a candidate for universal thresholding. DE is employed as an optimizer in the mentioned minimization exercise. By this way, the thresholding task is changed to an optimization problem. In order to solve this problem, the DE is employed with a very small population size ($N_p = 5$), called micro-DE. A small population size results a shorter computation time which is a crucial factor for the image processing tasks to make them suitable for online applications (e.g., robotics or production line control).

After comprehensive evaluation of more than 40 image thresholding techniques, Sezgin and Sankur [2004] concluded that the Kittler [Sezan 1985] is the best overall performing method. For this reason, and like many other thresholding works [Tizhoosh 2005a], the proposed method is compared with the Kittler. In the final part of this chapter, as a case study, the micro-DE and micro-ODE (micro-DE equipped with opposition-based initialization) are compared in terms of convergence speed and robustness.

Organization of this chapter is as follows: The proposed image thresholding approach is presented in section 6.2. Experimental investigation is presented in section 6.3. The micro-DE and micro-ODE are compared in section 6.4. Finally, the chapter is summarized

and concluded in section 6.5.

6.2 Proposed Image Thresholding Approach

When we are comparing the input grey-level image and corresponding thresholded version we perceptually map darker pixels in the input image to the black pixels in the thresholded image and lighter ones to the white pixels. With this method, we subjectively measure the quality of the thresholding. The same procedure happens even for a person who knows nothing about image processing concepts. We will have a high quality thresholded image when the mentioned similarity is high, or in other words, the dissimilarity is low. So, the thresholding task can be understood as an optimization problem. Before describing the new approach, the objective function for this optimization exercise should be defined.

Objective function - For an $M \times N$ input grey-level image, I (normalized in $[0, 1]$), and the corresponding thresholded image, $B(T) \in \{0, 1\}$, with the threshold value T, the objective function $f(T)$ is defined as follows:

$$f(T) = \sum_{i=1}^{M} \sum_{j=1}^{N} |I_{ij} - B(T)_{ij}|. \tag{6.1}$$

Minimization of this objective function means minimizing the dissimilarity between the input image and the thresholded image (see Figure 6.1).

In order to solve this one-dimensional minimization problem, the DE with very small population size ($N_p = 5$), micro-DE, is utilized. The pseudo-code representation of the proposed thresholding approach is shown in Algorithm 4.

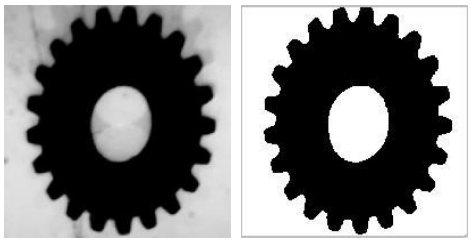


Fig. 6.1: A sample image to show that darker and lighter pixels in input grey-level image are matchable with black and white pixels in the thresholded image. In order to maximize the matching level, the dissimilarity between these two images, $f(T)$, should be minimized.

Algorithm 4 Proposed Thresholding Approach (micro-DE version)

1: Random population initialization, P_0
2: Calculate objective value (dissimilarity measure) for each individual in the population //DE's evolution steps
3: **while** (satisfying termination criteria) **do**
4: Mutation
5: Crossover
6: Selection
7: **end while**
8: Thresholding input image with the found optimal value of thresholding level, T_{op}

6.3 Experimental Verifications

In order to investigate the performance of the new approach, 16 hard-to-threshold images were selected; all images are frequently used in the image processing literature [Tizhoosh 2005a; Sezgin and Sankur 2004].

The following micro-DE control parameters are set for all conducted experiments with no attempt to achieve their optimal values.

- Population size, $N_p = 5$
- Differential amplification factor, $F = 0.9$

- Crossover probability constant, $C_r = 0.9$
- Strategy: DE/rand/1/bin
- Maximum function calls, NFC_{MAX}=200 (300 for image no. 9)

Threshold results of applying the proposed approach are presented in Table 6.1. As shown, also corresponding ground-truth image (created manually) and the result of the Kittler method are given for comparison. By visual evaluation, the new approach, at least over ten of the images (image no. 1,3,4,6,7,8,11,13,15,16) shows better results than Kittler.

For all images, plot of objective function $f(T)$ versus thresholding value T is presented in Figures 6.2 and 6.3. Furthermore, the result for the different thresholding values is given on each curve. Asymmetric shapes, flat surfaces, and also steep edges in the plotted graphs show that the objective function can be challenging although it is a one-dimensional problem.

A wide range of image quality measures have been proposed by image processing researchers [Winkler 2000; Wang and Bovik 2002; Martens and Meesters 1998]. In this section, results of Kittler and the proposed approach are compared by reference-based objective assessment. Reference or ground-truth images have been manually prepared to serve as gold/ideal thresholded image for each test image. To compare two binary images, Misclassification Error (ME) [Sezgin and Sankur 2004; Yasnoff et al. 1977] can be a reasonable and straightforward measure to use. It calculates the percentage of foreground pixels which have been assigned wrongly to background and vice versa:

$$ME = \frac{|B_O \cap F_T| + |F_O \cap B_T|}{|B_O| + |F_O|}, \tag{6.2}$$

where B_O, F_O, B_T, and F_T are the background and foreground pixels of the ground-truth image and the background and foreground pixels of the test image, respectively. $|\cdot|$ denotes the cardinality of the set.

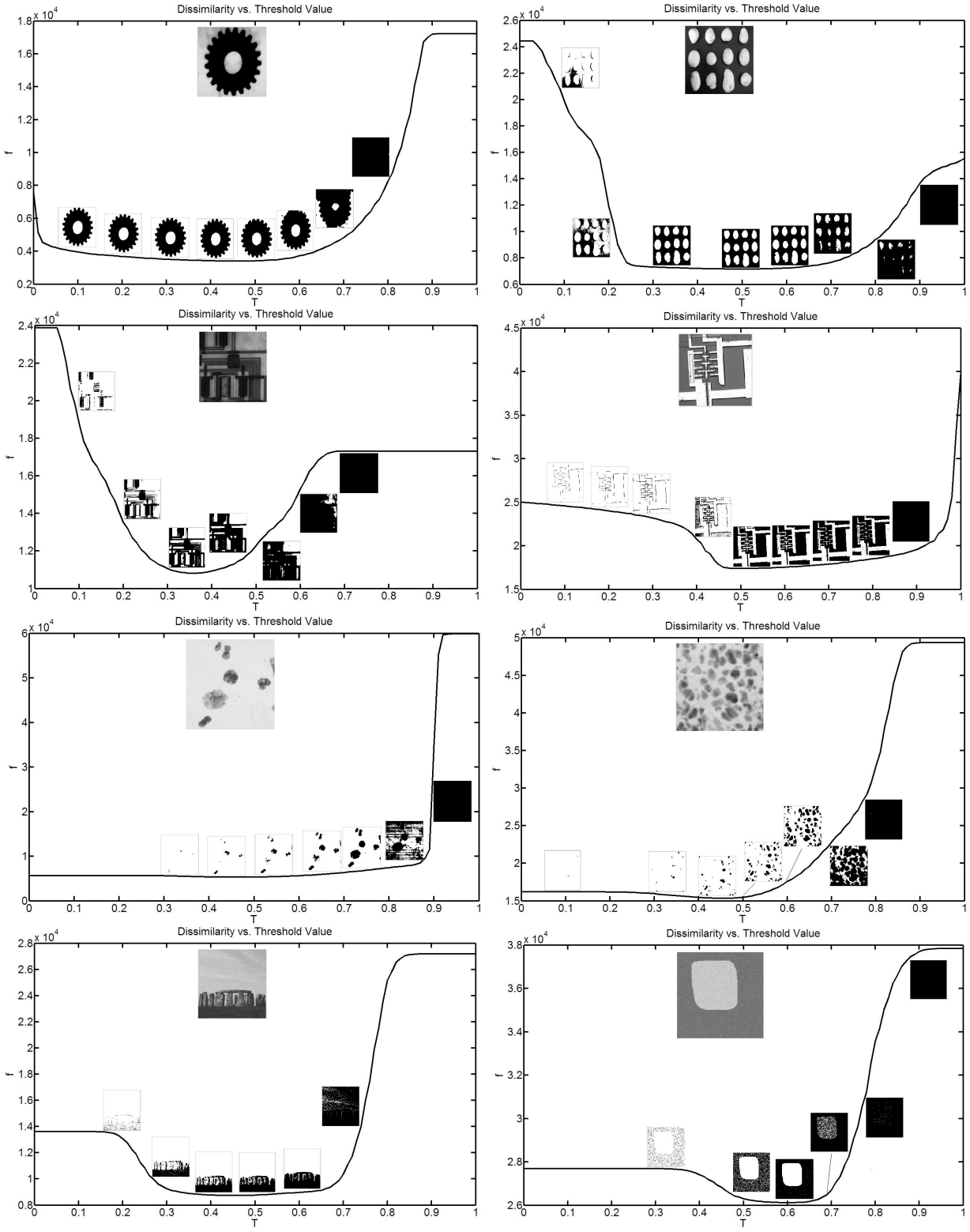


Fig. 6.2: Graphical illustration of dissimilarity (objective function, $f(T)$) vs. thresholding value (T). The thresholding results are presented on the curves.

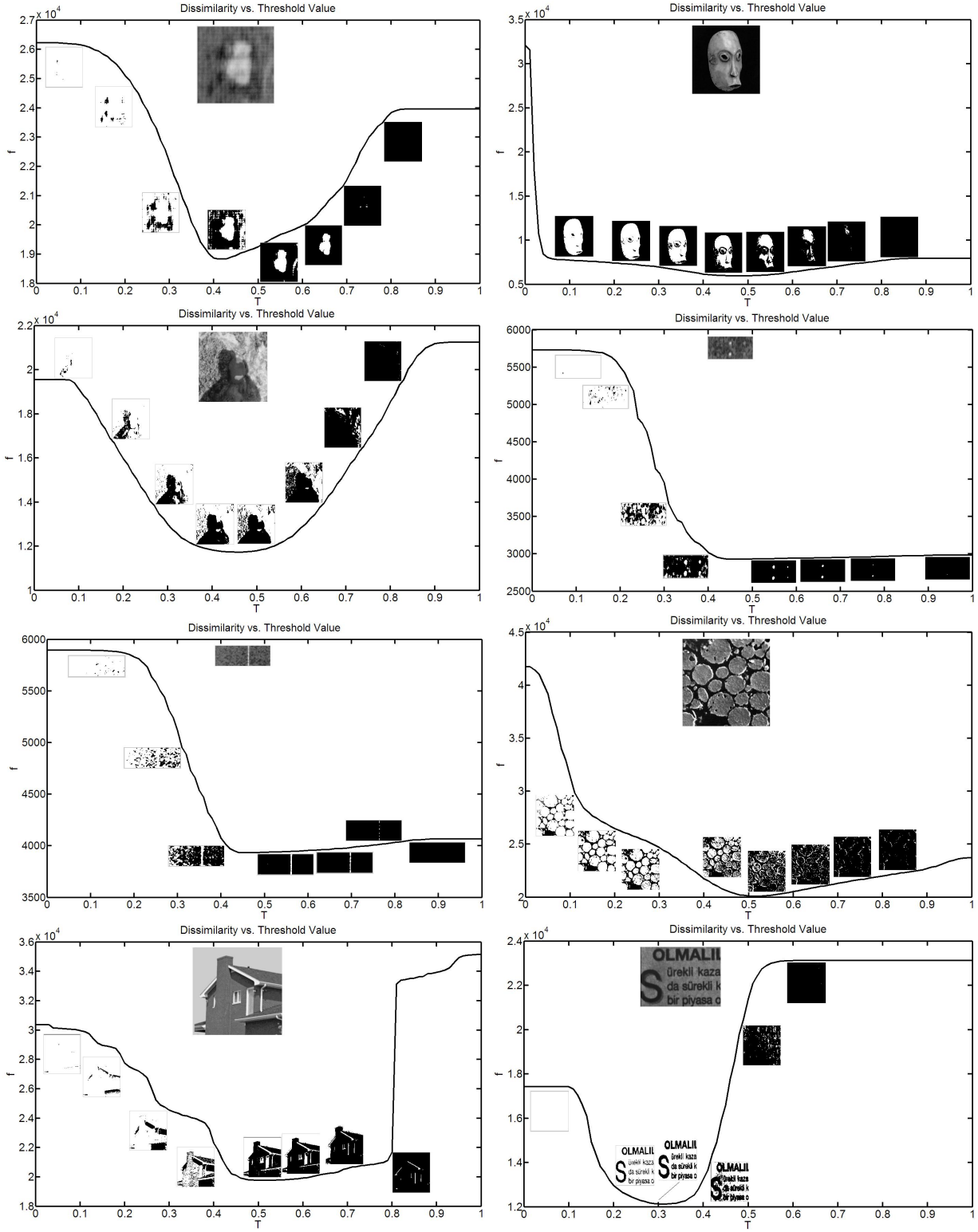
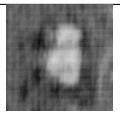
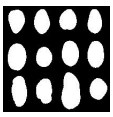
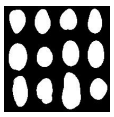
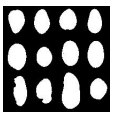
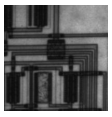
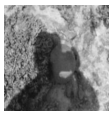
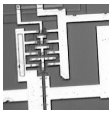
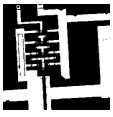
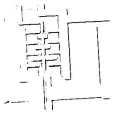
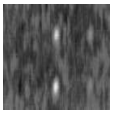
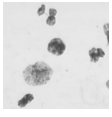
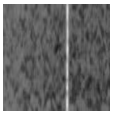


Fig. 6.3: Continued from Figure 6.3.

Table 6.1: Thresholding results. Input image, corresponding manually created ground-truth (gold) image, result of Kittler method, and result of the proposed approach (micro-DE).

no.	Image	Gold	Kittler	micro-DE	no.	Image	Gold	Kittler	micro-DE
1					9				
2					10				
3					11				
4					12				
5					13				
6					14				
7					15				
8					16				

By utilizing this error measure, the similarity index η can be defined as follows:

$$\eta = (1 - ME) \times 100\%. \tag{6.3}$$

Table 6.2 summarizes the results of objective assessment for 16 test images. Results of the Kittler and new method are compared with the gold image. The best result, in each case has been indicated in boldface. According to the mentioned similarity index, the Kittler shows better results for 5 cases; micro-DE performs better for 10 cases; and for one case the results is almost the same (results with difference less than 1% are considered the same).

Table 6.2: Results of objective assessment for 16 test images. The best result in each case has been highlighted in boldface. η is the similarity index.

no.	η_K	$\eta_{\text{micro-DE}}$	no.	η_K	$\eta_{\text{micro-DE}}$
1	89.88	**97.85**	9	**93.47**	82.45
2	**99.44**	98.40	10	**98.32**	84.48
3	77.84	**79.85**	11	78.19	**91.18**
4	41.49	**96.14**	12	99.83	99.57
5	**92.83**	90.05	13	28.14	**99.25**
6	52.98	**57.78**	14	**97.16**	49.76
7	81.80	**93.87**	15	48.68	**97.97**
8	72.78	**99.68**	16	83.07	**98.56**

6.4 Comparison of micro-DE and micro-ODE

In order to accelerate micro-DE, it is equipped with opposition-based initialization (micro-ODE). Because of the very small population size and also small number of required function calls to solve the objective function, the opposition-based generation jumping is not embedded in micro-ODE. Algorithm 5 presents pseudo-code for micro-ODE. The only difference between micro-DE and micro-ODE is on the population initialization. The first one uses the random initialization and the second one utilizes the opposition-based initialization.

The minimum values for the objective function, f_{min}, for each image (kept from the

Algorithm 5 Proposed Thresholding Approach (micro-ODE version)

Random population initialization, P_0
for $i = 0$ to N_p **do**
  for $j = 0$ to D **do**
    $OP_{0i,j} \leftarrow a_j + b_j - P_{0i,j}$
  end for
end for
Select N_p fittest individuals from set the $\{P_0, OP_0\}$ as initial population P_0
Calculate objective value (dissimilarity measure) for each individual in the population //DE's evolution steps
while (satisfying termination criteria) **do**
  Mutation
  Crossover
  Selection
end while
Thresholding input image with the found optimal value of thresholding level, T_{op}

micro-DE experiments), is used as value to reach (VTR) to compare convergence rate and robustness of micro-DE and micro-ODE. All control parameters are the same as before. The results of 100 trials per image for both algorithms are summarized in Table 6.3. Micro-ODE performs faster for 13 images. Both algorithms have the same NFCs for two images. For thresholding of 16 images, micro-DE needs 875 function calls while this number is 761 for micro-ODE (13% convergence rate improvement). The success rate is almost the same for both (0.98 vs. 0.99).

Table 6.3: Comparison of micro-DE and micro-ODE. Reported NFCs are the average over 100 trials for each image. The smaller NFC in each case has been indicated in boldface.

		micro-DE		micro-ODE	
no.	VTR=f_{min}	*NFC*	*SR*	*NFC*	*SR*
1	3402	45	1	**36**	0.98
2	7160	44	1	**37**	1
3	10820	47	1	**34**	1
4	17385	41	0.98	**29**	1
5	5321	84	0.94	**58**	1
6	15335	77	1	**75**	0.94
7	8727	74	0.94	**64**	0.98
8	26117	58	1	**54**	1
9	18825	25	1	**21**	1
10	5944	40	1	**30**	1
11	11730	62	1	62	1
12	2929	43	1	43	1
13	3932	40	1	**36**	1
14	20088	**64**	0.98	66	0.98
15	19761	59	0.98	**50**	1
16	12123	72	1	**66**	0.98
Overall		875	0.98	**761**	**0.99**

6.5 Summary

In this chapter, an optimization-based image thresholding approach has been introduced. micro-DE segmented the image into two classes by minimizing the dissimilarity between input grey-level image and binary (thresholded) image. The proposed approach was compared with a well-known method, the Kittler, through an objective assessment. Results confirmed that the proposed approach is superior to the Kittler algorithm over the selected test set.

The most important part of the proposed approach is the definition of the objective

function. As seen, micro-DE, as an optimizer, minimizes the dissimilarity between grey-level image and thresholded image. This dissimilarity is measured by pixel-by-pixel comparison of the binary and normalized grey-level images. The main drawback is that employing an evolutionary algorithm (DE) to threshold image shows a higher computational time. Employing the DE with a small population size (micro-DE) was in this direction to make computation time shorter. Furthermore, micro-DE was accelerated 13% by embedding opposition-based population initialization while the success rate remaind the same.

Chapter 7

Conclusions and Future Work

Reasoning draws a conclusion, but does not make the conclusion certain, unless the mind discovers it by the path of experience.

– Roger Bacon

Most creative work is a process of people passing ideas and inspirations from the past into the future and adding their own creativity along the way.

– Joichi Ito

7.1 Conclusions

Finding more accurate solution(s) in a shorter period of time for complex problems is the main goal of all evolutionary algorithms. Although the opposition concept has a very old history in other sciences, that is the first time that this concept is employed to enhance population-based algorithms. Conclusions for this attempt can be summarized as follows:

Results are promising, however, the opposition-based optimization is still in its infancy. Results confirm that the opposition concept has the potential to play a positive role in optimization. But, it is important to mention that the current study constitutes the first step of this newly opened direction. Like many other new concepts, opposition-based optimization needs further studies to shed light on its benefits, weaknesses, and limitations. In fact, the main claim is not defeating DE or any of its numerous versions but to introduce a new notion into optimization via metaheuristics; this is the notion of opposition.

As a key point, the performance of the opposition-based optimization is directly depended on how often and how many opposite points are evaluated. Any individual evaluation costs a function call. So, a smartly controlled opposition concept is highly advised to prevent overwhelming of the acceleration benefits by the extra computation load for evaluating opposite points.

Generally speaking, the high convergence speed and robustness are two conflicting objectives. A trade-off between the fastness and robustness is essential in evolutionary optimization. These are very similar to the exploration and exploitation. Although, there is no rigid boundary between them. The optimal division of the search time between these two steps is a challenging task and problem-oriented.

The optimal control parameters are problem-oriented such that developing a self-adaptive or adaptive algorithm is a valuable attempt. Many studies confirm that for population-based algorithms the optimal parameters are problem-oriented. Running limited trials to

determine desirable parameters is a common approach (if not a practical way for time consuming objective functions). For this reason, the self-adaptive/adaptive control parameter setting would be a valuable improvement.

Introducing an extra control parameter is not appreciated. For ODE, the jumping rate is added to DE's three parameters. Having more control parameters makes the optimal parameter setting and analysis harder.

Higher jumping rates are not recommended because it causes the fast decreasing of the population diversity and so premature convergence, in particular for the multimodal problems. Indeed, the generation jumping makes the exploration time shorter which is directly caused by the jumping rate. For high dimensional problems with small population size a rather small jumping rate is recommended.

The benefits of opposition-based optimization are not the same for different problems. This is due to using fix settings for the parameters instead of the optimal ones and/or the different characteristics of each problem (e.g., modality, dimension, surface features, separability of the variables and so on). Similar to all optimization approaches, ODE does not present a consistent behavior over different problems. However, in overall and over the employed benchmark test suite, ODE performed better than classical DE and five other evolutionary optimizers.

The proposed opposition-based schemes are general enough to be applied to other population - based algorithms. The opposition-based schemes work at the population level and leave the evolutionary part of the algorithms untouched. This generality gives higher flexibility to these schemes to be embedded inside other population-based algorithms for further investigation.

Standard benchmark test suite and also benchmarking methodologies are required. Without any unified test platform and comparison methodology, the concrete judgment among the competitors is virtually impossible. This shortage can be addressed by the optimization community. Although, figuring out all aspects of an optimizer by applying it on one test

suite (even a very comprehensive one), seems impossible, it nevertheless supports unified comparative studies.

7.2 Contributions

The author first started to work on image processing [Rahnamayan et al. 2005a,b,c,d,e,f,g, 2006a,h; Kachouie et al. 2006] but when he was faced with computational time problem of the population-based algorithms, he focused on accelerating them and finally his studies has led to the following contributions.

- A general scheme of opposition-based population initialization is proposed to accelerate evolutionary algorithms [Rahnamayan et al. 2006b; Rahnamayan 2006]. For the first time the opposition concept was employed to accelerate an optimizer.

- A framework for the opposition-based evolutionary algorithm is suggested to accelerate the convergence rate [Rahnamayan et al. 2006g]. The opposition-based generation jumping is employed to achieve a higher acceleration rate.

- The opposition-based differential evolution (ODE) is introduced [Rahnamayan et al. 2006d,e] and a comprehensive set of experiments is conducted to verify its efficiency. The thesis main contribution will appear in ***IEEE Transactions on Evolutionary Computation*** [Rahnamayan et al. 2006f].

- As the thesis' second major contribution, mathematical proofs confirming experimental verifications are presented [Rahnamayan et al. 2006c, 2007b] to show why opposite numbers are beneficial. The achieved results could be valuable not only for the optimization area but also in the machine learning or soft computing research areas.

- A new image thresholding approach is proposed and it was compared with the Kittler method. The proposed approach outperforms Kittler method in overall. Furthermore, the results confirm that the micro-ODE is faster than micro-DE because of opposition-based population initialization.

- The opposition-based DE with variable jumping rate (ODEVJR) is suggested to enhance ODE's performance [Rahnamayan et al. 2007a] (see Appendix B). In the first version of ODE, a constant value for jumping rate is used. For the ODEVJR, two types of time variable jumping rate are investigated (linearly increasing and decreasing functions). Results confirm that a higher jumping rate is more desirable during the exploration than during the exploitation phase.

- The quasi-oppositional DE (QODE) is introduced as a potential extension for ODE [Rahnamayan et al. 2007c] (see Appendix B). The QODE employs exactly the same schemes of ODE for population initialization and generation jumping; just instead of opposite the quasi-opposite individuals are utilized. Furthermore, the mathematical proofs are provided to support the current extension.

- DE is enhanced by local tuning of the fittest individual. A local search approach has been embedded inside the classical DE. By this way, the method gives an extra chance to local improvement of the best individual in the current population [Jonasson and Rahnamayan 2006].

7.3 Future Work

OBO opens a fresh perspective in the population-based algorithms to accelerate optimization process. Because of this novelty, many studies can be conducted to extend, enhance, or apply the proposed schemes and algorithms. Some of these directions are as follows:

- *Generalizing ODE to handle constrained functions and multi-objective optimization* - For most practical applications, we are faced with constraint functions and also with multi-objective problems. So far, there are many approaches for handling constraints in DE and also for multi-objective optimization using DE. All of these proposals can be borrowed and investigated to generalize ODE to solve multi-objective constrained problems.

- *Employing proposed opposition-based schemes to accelerate DE's enhanced versions or other population-based algorithms* - The proposed opposition-based schemes for population initialization and generation jumping are general enough and can be utilized to accelerate DE's extended versions (such as self-adaptive DE [Brest et al. 2006], fuzzy adaptive DE [Liu and Lampinen 2005]) or other population-based algorithms (e.g., GAs and PSO). However, the introduction of the opposition-based version of other algorithms still remains widely open to research (such as the opposition-based continuous GA (CGA) or opposition-based free search (FS) [Penev and Littlefair 2005]).

- *Extending opposition-based optimization to non-continuous (discrete/interger) spaces* - Continuous opposition-based optimization was considered in this study. One obvious extension can be working with discrete variables instead of continuous ones. As it was mentioned before, DE (and similarly ODE) can work with discrete variables easily, but further experiments are required to figure out the performance of ODE on these sort of problems (e.g., combinatorial problems).

- *Developing adaptive/self-adaptive ODE* - Adaptive setting of the control parameters for any optimizer is widely appreciated. The proposed idea in [Brest et al. 2006] for self-adaptive setting of C_r and F can be used to set J_r.

The opposition-based optimization is simple to implement and open to be used in many different ways for different purposes. This study is a starting point in this direction

to confirm the potentials and motivate other researchers in optimization and machine learning fields to engage with the opposition concept.

Appendix A

List of Bound Constrained Global Optimization Test Functions

All algorithms that search for an extremum of an objective function perform exactly the same, according to any performance measure, when averaged over all possible objective functions.

– David H. Wolpert and William G. Macready in "No-free-lunch theorem"

- 1^{st} *De Jong* [Onwubolu and Babu 2004]

$$f_1(X) = \sum_{i=1}^{n} {x_i}^2,$$
$$\text{with} -5.12 \leq x_i \leq 5.12,$$
$$min(f_1) = f_1(0, ..., 0) = 0.$$

Unimodal, scalable, convex, and easy function.

- *Axis Parallel Hyper-Ellipsoid*

$$f_2(X) = \sum_{i=1}^{n} i{x_i}^2,$$
$$\text{with} -5.12 \leq x_i \leq 5.12,$$
$$min(f_2) = f_2(0, ..., 0) = 0.$$

Unimodal, scalable, convex, and easy function.

- *Schwefel's Problem 1.2*

$$f_3(X) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2,$$
$$\text{with} -65 \leq x_i \leq 65,$$
$$min(f_3) = f_3(0, ..., 0) = 0.$$

Unimodal and scalable function.

- *Rosenbrock's Valley* [Moré et al. 1981]

$$f_4(X) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right],$$
$$\text{with} -2 \leq x_i \leq 2,$$
$$min(f_4) = f_4(1, ..., 1) = 0.$$

This function is known as the Banana function. It is a non-convex unimodal classic optimization problem. Locating the minimum is very challenging for many optimizers, because the optimum is inside a long, narrow, parabolic shaped flat valley.

- *Rastrigin's Function* [Storn and Price 1997b]

$$f_5(X) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)),$$
$$\text{with} -5.12 \leq x_i \leq 5.12,$$
$$min(f_5) = f_5(0, ..., 0) = 0.$$

This function is highly multimodal and the location of the minima is regularly distributed.

- *Griewangk's Function* [Onwubolu and Babu 2004]

$$f_6(X) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1,$$
$$\text{with} -600 \leq x_i \leq 600,$$
$$min(f_6) = f_6(0, ..., 0) = 0.$$

This function has many regularly distributed local minima and hard to locate global minimum.

- *Sum of Different Power*

$$f_7(X) = \sum_{i=1}^{n} |x_i|^{(i+1)},$$
$$\text{with} -1 \leq x_i \leq 1,$$
$$min(f_7) = f_7(0, ..., 0) = 0.$$

This is a unimodal scalable function.

- *Ackley's Problem* [Storn and Price 1997b]

$$f_8(X) = -20 \exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}\right) - \exp\left(\frac{\sum_{i=1}^{n} \cos(2\pi x_i)}{n}\right) + 20 + e,$$
$$\text{with} -32 \leq x_i \leq 32,$$
$$min(f_8) = f_8(0, ..., 0) = 0.$$

The number of local minima is unknown.

- *Beale Function*

$$f_9(X) = [1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2 + [2.625 - x_1(1 - x_2^3)]^2,$$
$$\text{with} -4.5 \leq x_i \leq 4.5,$$
$$min(f_9) = f_9(3, 0.5) = 0.$$

- *Colville Function*

$$f_{10}(X) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1),$$
$$\text{with} -10 \leq x_i \leq 10,$$
$$min(f_{10}) = f_{10}(1, 1, 1, 1) = 0.$$

- *Easom Function* [Michalewicz 1996]

$$f_{11}(X) = -\cos(x_1)\cos(x_2)\exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2),$$
$$\text{with} -100 \leq x_i \leq 100,$$
$$min(f_{11}) = f_{11}(\pi, \pi) = -1.$$

This function is unimodal and the global minimum lays in a narrow area relative to the search space.

- *Hartmann Function 1* [Dixon and Szegö 1978]

$$f_{12}(X) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2\right),$$
$$\text{with } 0 \leq x_i \leq 1,$$
$$min(f_{12}) = f_{12}(0.114614, 0.555649, 0.852547) = -3.86278.$$

where:

$$\alpha = \begin{bmatrix} 1 & 1.2 & 3 & 3.2 \end{bmatrix},\ A = \begin{bmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix},\ P = \begin{bmatrix} 0.36890 & 0.11700 & 0.26730 \\ 0.46990 & 0.43870 & 0.74700 \\ 0.10910 & 0.87320 & 0.55470 \\ 0.03815 & 0.57430 & 0.88280 \end{bmatrix}.$$

It has four local minima.

- *Hartmann Function 2* [Dixon and Szegö 1978]

$$f_{13}(X) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{6} B_{ij}(x_j - Q_{ij})^2\right),$$
$$\text{with } 0 \leq x_i \leq 1,$$

$$min(f_{13}) = f_{13}(0.20169, 0.150011, 0.476874, 0.275332, 0.311652, 0.6573) = -3.32237.$$

where:

$$\alpha = \begin{bmatrix} 1 & 1.2 & 3 & 3.2 \end{bmatrix},\ B = \begin{bmatrix} 10 & 3 & 17 & 3.05 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix},$$

$$Q = \begin{bmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{bmatrix}.$$

It has four local minima.

• *Six Hump Camel Back Function* [Dixon and Szegö 1978; Michalewicz 1996]

$$f_{14}(X) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4,$$
$$\text{with} -5 \leq x_i \leq 5,$$
$$min(f_{14}) = f_{14}(0.0898, -0.7126)/(-0.0898, 0.7126) = -1.0316285.$$

This problem has six local minima, two of them are global.

• *Levy Function*

$$f_{15}(X) = \sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2(1 + \sin^2(3\pi x_{i+1})) + (x_n - 1)(1 + \sin^2(2\pi x_n)),$$
$$\text{with} -10 \leq x_i \leq 10,$$
$$min(f_{15}) = f_{15}(1, ..., 1) = 0.$$

This function has approximately 15^n local minima.

• *Matyas Function*

$$f_{16}(X) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2,$$
$$\text{with} -10 \leq x_i \leq 10,$$
$$min(f_{16}) = f_{16}(0, 0) = 0.$$

This function is unimodal.

• *Perm Function*

$$f_{17}(X) = \sum_{k=1}^{n}\left[\sum_{i=1}^{n}(i^k + 0.5)((\frac{1}{i}x_i)^k - 1)\right]^2,$$
$$\text{with} -n \leq x_i \leq n,$$
$$min(f_{17}) = f_{17}(1, 2, 3, ..., n) = 0.$$

This function is unimodal.

• *Michalewicz Function* [Onwubolu and Babu 2004]

$$f_{18}(X) = -\sum_{i=1}^{n} \sin(x_i)(sin(ix_i^2/\pi))^{2m},$$
$$\text{with } 0 \leq x_i \leq \pi, m = 10,$$

$min(f_{18_{(n=2)}}) = -1.8013, min(f_{18_{(n=5)}}) = -4.687658, min(f_{18_{(n=10)}}) = -9.66015.$

This function is multimodal and more difficult for larger m.

• *Zakharov Function*

$$f_{19}(X) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^4,$$
$$\text{with } -5 \leq x_i \leq 10,$$
$$min(f_{19}) = f_{19}(0, ..., 0) = 0.$$

This function is unimodal.

• *Branin Problem* [Dixon and Szegö 1978]

$$f_{20}(X) = a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1 - f)\cos(x_1) + e,$$
$$\text{with } -5 \leq x_1 \leq 10, 0 \leq x_2 \leq 15$$
$$\text{where } a = 1, b = \frac{5.1}{4\pi^2}, c = \frac{5}{\pi}, d = 6, e = 10, f = \frac{1}{8\pi}$$
$$min(f_{20}) = f_{20}(-\pi, 12.275)/(-\pi, 2.275)/(9.42478, 2.475) = 0.3979.$$

It has three minima.

- *Schwefel's Problem 2.22*

$$f_{21}(X) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|,$$
$$\text{with } -10 \leq x_i \leq 10,$$
$$min(f_{21}) = f_{21}(0, ..., 0) = 0.$$

This function is unimodal.

- *Schwefel's Problem 2.21*

$$f_{22}(X) = \max_i \{|x_i|, 1 \leq i \leq n\},$$
$$\text{with } -100 \leq x_i \leq 100,$$
$$min(f_{22}) = f_{22}(0, ..., 0) = 0.$$

This function is unimodal.

- *Step Function*

$$f_{23}(X) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2,$$
$$\text{with } -100 \leq x_i \leq 100,$$
$$min(f_{23}) = f_{23}(-0.5 \leq x_i < 0.5) = 0.$$

This function has steep edges and flat surfaces which makes optimizers prone to get stuck on those areas.

- *Noisy Quartic Function*

$$f_{24}(X) = \sum_{i=1}^{n} ix_i^4 + random[0,1),$$
$$\text{with } -1.28 \leq x_i \leq 1.28,$$
$$min(f_{24}) = f_{24}(0,...,0) = 0.$$

This is an easy unimodal function which is padded with uniform noise.

- *Kowalik's Function*

$$f_{25}(X) = \sum_{i=1}^{11} \left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right]^2,$$
$$\text{with } -5 \leq x_i \leq 5,$$
$$min(f_{25}) = f_{25}(0.19, 0.19, 0.12, 0.14) = 0.0003075.$$

where: a=[0.1957 0.1947 0.1735 0.1600 0.0844 0.0627 0.0456 0.0342 0.0323 0.0235 0.0246], b^{-1}=[0.25 0.5 1 2 4 6 8 10 12 14 16].

This function is multimodal (with a few local minima).

- *Shekel 5 Problem* [Dixon and Szegö 1978]

$$f_{26}(X) = -\sum_{i=1}^{5} \frac{1}{\sum\limits_{j=1}^{4} (x_j - a_{ij})^2 + c_i},$$

where

$$a_{ij} = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \end{bmatrix},$$

$$c_i = \begin{bmatrix} 0.1 & 0.2 & 0.2 & 0.4 & 0.4 \end{bmatrix},$$

with $0 \leq x_j \leq 10$,

$$min(f_{26}) = f_{26}(4, 4, 4, 4) \approx -10.1499.$$

The number of local minima is five.

- *Shekel 7 Problem* [Dixon and Szegö 1978]

$$f_{27}(X) = -\sum_{i=1}^{7} \frac{1}{\sum_{j=1}^{4} (x_j - a_{ij})^2 + c_i},$$

where

$$a_{ij} = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \end{bmatrix},$$

$$c_i = \begin{bmatrix} 0.1 & 0.2 & 0.2 & 0.4 & 0.4 & 0.6 & 0.3 \end{bmatrix},$$

with $0 \leq x_j \leq 10$,

$$min(f_{27}) = f_{27}(4, 4, 4, 4) \approx -10.3999.$$

The number of local minima is seven.

- *Shekel 10 Problem* [Dixon and Szegö 1978]

$$f_{28}(X) = -\sum_{i=1}^{10} \frac{1}{\sum_{j=1}^{4} (x_j - a_{ij})^2 + c_i},$$

where

$$a_{ij} = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{bmatrix},$$

$$c_i = \begin{bmatrix} 0.1 & 0.2 & 0.2 & 0.4 & 0.4 & 0.6 & 0.3 & 0.7 & 0.5 & 0.5 \end{bmatrix},$$

with $0 \leq x_j \leq 10$,

$$min(f_{28}) = f_{28}(4, 4, 4, 4) \approx -10.5319.$$

The number of local minima is ten.

- *Tripod Function*

$$f_{29}(X) = p(x_2)(1 + p(x_1)) + |(x_1 + 50p(x_2)(1 - 2p(x_1)))| + |(x_2 + 50(1 - 2p(x_2)))|,$$

with $-100 \leq x_i \leq 100$,

$$min(f_{29}) = f_{29}(0, -50) = 0,$$

where $p(x) = 1$ for $x \geq 0$ otherwise $p(x) = 0$.

This function is a non-continuous multimodal function. Optimizers are prone to be trapped in its two local minima.

- 4^{th} *De Jong* [Onwubolu and Babu 2004]

$$f_{30}(X) = \sum_{i=1}^{n} i{x_i}^4,$$
$$\text{with} -1.28 \leq x_i \leq 1.28,$$
$$min(f_{30}) = f_{30}(0, ..., 0) = 0.$$

This is an easy unimodal function.

- *Alpine Function*

$$f_{31}(X) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1x_i|,$$
$$\text{with} -10 \leq x_i \leq 10,$$
$$min(f_{31}) = f_{31}(0, ..., 0) = 0.$$

This function is multimodal and not symmetrical.

- *Schaffer's Function 6*

$$f_{32}(X) = 0.5 + \frac{sin^2\sqrt{(x_1^2 + x_2^2)} - 0.5}{1 + 0.01(x_1^2 + x_2^2)^2},$$
$$\text{with} -10 \leq x_i \leq 10,$$
$$min(f_{32}) = f_{32}(0, 0) = 0.$$

This function is multimodal.

• *Pathological Function* [Onwubolu and Babu 2004]

$$f_{33}(X) = \sum_{i=1}^{n-1} \left(0.5 + \frac{sin^2\sqrt{(100x_i^2 + x_{i+1}^2)} - 0.5}{1 + 0.001(x_i^2 - 2x_i x_{i+1} + x_{i+1}^2)^2} \right),$$
$$\text{with} -100 \leq x_i \leq 100,$$
$$min(f_{33}) = f_{33}(0, ..., 0) = 0.$$

This function is multimodal and extremely complex.

• *Inverted Cosine Wave Function (Masters)* [Onwubolu and Babu 2004]

$$f_{34}(X) = -\sum_{i=1}^{n-1} \left(\exp\left(\frac{-(x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1})}{8} \right) \cos\left(4\sqrt{x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1}} \right) \right),$$
$$\text{with} -5 \leq x_i \leq 5,$$
$$min(f_{34}) = f_{34}(0, ..., 0) = -n + 1.$$

This function is multimodal.

• *Aluffi-Pentini's Problem* [Aluffi-Pentini et al. 1985]

$$f_{35}(X) = 0.25x_1^4 - 0.5x_1^2 + 0.1x_1 + 0.5x_2^2,$$
$$\text{with} -10 \leq x_i \leq 10,$$
$$min(f_{35}) = f_{35}(-1.0465, 0) = -0.3523.$$

The number of local minima is two.

• *Becker and Lago Problem* [Price 1977]

$$f_{36}(X) = (\mid x_1 \mid -5)^2 + (\mid x_2 \mid -5)^2,$$
$$\text{with} -10 \leq x_i \leq 10,$$
$$min(f_{36}) = f_{36}(\pm 5, \pm 5) = 0.$$

This function has four minima.

- *Bohachevsky 1 Problem* [Bohachevsky et al. 1986]

$$f_{37}(X) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7,$$
$$\text{with} -50 \leq x_i \leq 50,$$
$$min(f_{37}) = f_{37}(0,0) = 0.$$

The number of local minima is not known.

- *Bohachevsky 2 Problem* [Bohachevsky et al. 1986]

$$f_{38}(X) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3,$$
$$\text{with} -50 \leq x_i \leq 50,$$
$$min(f_{38}) = f_{38}(0,0) = 0.$$

The number of local minima is unknown.

- *Camel Back-3 Three Hump Problem* [Dixon and Szegö 1975]

$$f_{39}(X) = 2x_1^2 - 1.05x_1^4 + \frac{1}{6}x_1^6 + x_1x_2 + x_2^2,$$
$$\text{with} -5 \leq x_i \leq 5,$$
$$min(f_{39}) = f_{39}(0,0) = 0.$$

The number of local minima is three.

- *Dekkers and Aarts Problem* [Dekkers and Aarts 1991]

$$f_{40}(X) = 10^5 x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-5}(x_1^2 + x_2^2)^4,$$
$$\text{with} -20 \leq x_i \leq 20,$$
$$min(f_{40}) = f_{40}(0, \pm 15) = -24777.$$

It has three local minima.

- *Exponential Problem* [Breiman and Cutler 1993]

$$f_{41}(X) = \exp\left(-0.5 \sum_{i=1}^{n} x_i^2\right),$$
$$\text{with} -1 \leq x_i \leq 1,$$
$$min(f_{41}) = f_{41}(0, ..., 0) = 1.$$

This is a unimodal function.

- *Goldstein and Price Problem* [Dixon and Szegö 1978]

$$f_{42}(X) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$$
$$\times[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)],$$
$$\text{with} -2 \leq x_i \leq 2,$$
$$min(f_{42}) = f_{42}(0, -1) = 3.$$

The number of local minima is four.

- *Gulf Research Problem* [Moré et al. 1981]

$$f_{43}(X) = \sum_{i=1}^{99} \left[\exp\left(-\frac{(u_i - x_2)^{x_3}}{x_1} \right) - 0.01 \times i \right]^2,$$
$$\text{where } u_i = 25 + [-50 \ln(0.01 \times i)]^{\frac{1}{1.5}}$$
$$\text{with } 0.1 \le x_1 \le 100, 0 \le x_2 \le 25.6, 0 \le x_3 \le 5,$$
$$min(f_{43}) = f_{43}(50, 25, 1.5) = 0.$$

- *Helical Valley Problem* [Wolfe 1978]

$$f_{44}(X) = 100 \left[(x_2 - 10\theta)^2 + \left(\sqrt{(x_1^2 + x_2^2)} - 1 \right)^2 \right] + x_3^2,$$
$$\text{where } \theta = \begin{cases} \frac{1}{2\pi} \arctan \frac{x_2}{x_1}, & \text{if } x_1 \ge 0 \\ \frac{1}{2\pi} \arctan \frac{x_2}{x_1} + \frac{1}{2}, & \text{if } x_1 < 0 \end{cases}$$
$$\text{with } -10 \le x_i \le 10,$$
$$min(f_{44}) = f_{44}(1, 0, 0) = 0.$$

This is a steep-sided valley which follows a spiral shaped path.

- *Hosaki Problem* [Benke and Skinner 1991]

$$f_{45}(X) = (1 - 8x_1 + 7x_1^2 - \frac{7}{3}x_1^3 + \frac{1}{4}x_1^4)x_2^2 \exp(-x_2),$$
$$\text{with } 0 \le x_1 \le 5, 0 \le x_2 \le 6,$$
$$min(f_{45}) = f_{45}(4, 2) \approx -2.3458.$$

The number of local minima is two.

- *Levy and Montalvo 1 Problem* [Levy and Montalvo 1985]

$$f_{46}(X) = \frac{\pi}{n}\left(10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i-1)^2[1+10sin^2(\pi y_{i+1})] + (y_n-1)^2\right),$$
$$\text{where } y_i = 1 + \frac{1}{4}(x_i+1),$$
$$\text{with } -10 \le x_i \le 10,$$
$$min(f_{46}) = f_{46}(-1,-1,...,-1) = 0.$$

The number of local minima is approximately 5^n.

- *McCormick Problem* [McCormick 1982]

$$f_{47}(X) = \sin(x_1+x_2) + (x_1-x_2)^2 - \frac{3}{2}x_1 + \frac{5}{2}x_2 + 1,$$
$$\text{with } -1.5 \le x_1 \le 4, -3 \le x_2 \le 3,$$
$$min(f_{47}) = f_{47}(-0.547,-1.547) \approx -1.9133.$$

The number of local minima is two.

- *Miele and Cantrell Problem* [Wolfe 1978]

$$f_{48}(X) = (\exp(x_1)-x_2)^4 + 100(x_2-x_3)^6 + \tan^4(x_3-x_4) + x_1^8,$$
$$\text{with } -1 \le x_i \le 1,$$
$$min(f_{48}) = f_{48}(0,1,1,1) = 0.$$

The number of local minima is unknown.

Table A.1: Data for Multi-Gaussian Problem

i	a_i	b_i	c_i	d_i
1	0.5	0.0	0.0	0.1
2	1.2	1.0	0.0	0.5
3	1.0	0.0	-0.5	0.5
4	1.0	-0.5	0.0	0.5
5	1.2	0.0	1.0	0.5

- *Multi-Gaussian Problem* [Benke and Skinner 1991]

$$f_{49}(X) = \sum_{i=1}^{5} a_i \exp\left(-((x_1 - b_i)^2 + (x_2 - c_i)^2)/d_i^2\right),$$
$$\text{with } -2 \leq x_i \leq 2,$$
$$min(f_{49}) = f_{49}(-0.01356, -0.01356) \approx 1.29695.$$

This function has five local minima and a saddle point.

- *Neumaier 2 Problem*

$$f_{50}(X) = \sum_{k=1}^{n} \left(b_k - \sum_{i=1}^{n} x_i^k \right)^2,$$
$$\text{for n=4 } b = (8, 18, 44, 114),$$
$$\text{with } 0 \leq x_i \leq n,$$
$$min(f_{50}) = f_{51}(1, 2, 2, 3) = 0.$$

This a unimodal function.

- *Odd Square Problem*

$$f_{51}(X) = -\left(1.0 + \frac{0.2d}{(D+0.1)}\right)\cos(D\pi)\exp\left(\frac{-D}{2\pi}\right),$$

$$\text{where } d = \sqrt{\sum_{i=1}^{n}(x_i - b_i)^2}, D = \sqrt{n}(max|x_i - b_i|),$$

$$\text{and } \underline{b} = (1, 1.3, 0.8, -0.4, -1.3, 1.6, -2, -6, 0.5, 1.4, 1,$$

$$1.3, 0.8, -4, -1.3, 1.6, -0.2, -0.6, 0.5, 1.4)$$

$$\text{with } -15 \leq x_i \leq 15$$

$$min(f_{51}) = f_{51}(\underline{b}) \approx -1.143833.$$

The number of local minima, as a function of n, is unknown. There are many solutions near $\underline{b}$.

- *Paviani Problem* [Himmelblau 1972]

$$f_{52}(X) = \sum_{i=1}^{10}\left[(\ln(x_i - 2))^2 + (\ln(10 - x_i))^2\right] - \left(\prod_{i=1}^{10} x_i\right)^{0.2},$$

$$\text{with } 2 \leq x_i \leq 10,$$

$$min(f_{52}) = f_{52}(9.351, 9.351, ..., 9.351) = -45.778.$$

This a unimodal function.

- *Periodic Problem* [Price 1977]

$$f_{53}(X) = 1 + \sin^2 x_1 + \sin^2 x_2 - 0.1exp(-x_1^2 - x_2^2),$$

$$\text{with } -10 \leq x_i \leq 10,$$

$$min(f_{53}) = f_{53}(0, 0) = 0.9.$$

The number of local minima is 50.

- *Powell's Quadratic Problem* [Wolfe 1978]

$$f_{54}(X) = (x_1 + 10x_1)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4,$$
$$\text{with } -10 \leq x_i \leq 10,$$
$$min(f_{54}) = f_{54}(0, 0, 0, 0) = 0.$$

This a unimodal function but it is difficult to find the minimum with higher accuracy.

- *Price's Transistor Modeling Problem* [Price 1977]

$$f_{55}(x_1, x_2, ..., x_9) = \gamma^2 + \sum_{k=1}^{4} \left(\alpha_k^2 + \beta_k^2\right),$$
$$\text{where}$$
$$\alpha_k = (1 - x_1 x_2) x_3 \{\exp[x_5(g_{1k} - g_{3k} x_7 \times 10^{-3} - g_{5k} x_8 \times 10^{-3})] - 1\} - g_{5k} + g_{4k} x_2,$$
$$\beta_k = (1 - x_1 x_2) x_4 \{\exp[x_6(g_{1k} - g_{2k} - g_{3k} x_7 \times 10^{-3} - g_{4k} x_9 \times 10^{-3})] - 1\} - g_{5k} x_1 + g_{4k},$$
$$\gamma = x_1 x_3 - x_2 x_4,$$
$$g_{ik} = \begin{bmatrix} 0.485 & 0.752 & 0.869 & 0.982 \\ 0.369 & 1.254 & 0.703 & 1.455 \\ 5.2095 & 10.0677 & 22.9274 & 20.2153 \\ 23.3037 & 101.779 & 111.461 & 191.267 \\ 28.5132 & 111.8467 & 134.3884 & 211.4823 \end{bmatrix},$$
$$\text{with } -10 \leq x_i \leq 10,$$
$$min(f_{55}) = f_{55}(0.9, 0.45, 1, 2, 8, 8, 5, 1, 2) = 0.$$

The number of local minima is unknown.

- *Salomon Problem* [Salomon 1995]

$$f_{56}(X) = 1 - \cos(2\pi \parallel x \parallel) + 0.1 \parallel x \parallel,$$

$$\text{where } \parallel x \parallel = \sqrt{\sum_{i=1}^{n} x_i^2},$$
$$\text{with } -100 \leq x_i \leq 100,$$
$$min(f_{56}) = f_{56}(0, 0, ..., 0) = 0.$$

The number of local minima, as a function of n, is unknown. Its landscape is like a pond with ripples.

- *Schaffer 2 Problem* [Michalewicz 1996]

$$f_{57}(X) = (x_1^2 + x_2^2)^{0.25}(\sin^2(50(x_1^2 + x_2^2)^{0.1}) + 1),$$
$$\text{with } -100 \leq x_i \leq 100,$$
$$min(f_{57}) = f_{57}(0, 0) = 0.$$

The number of local minima is unknown.

- *Wood's Function*

$$f_{58}(X) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 +$$
$$10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1),$$
$$\text{with } -10 \leq x_i \leq 10,$$
$$min(f_{58}) = f_{58}(1, 1, 1, 1) = 0.$$

This function has a saddle point.

Appendix B

Enhancement Directions for ODE

Non-violent resistance implies the very opposite of weakness. Defiance combined with non-retaliatory acceptance of repression from one's opponents is active, not passive. It requires strength, and there is nothing automatic or intuitive about the resoluteness required for using non-violent methods in political struggle and the quest for Truth.

– Mahatma Gandhi, 1936

The opposite of love is not hate, it's indifference.

– Elie Wiesel

B.1 Quasi-Oppositional Differential Evolution (QODE)

In this section, the effects of replacing opposite numbers with quasi-opposite numbers in the ODE are investigated. We call the new method QODE which employs exactly the same schemes of ODE for population initialization and generation jumping. As we know, the middle point of the search interval has the highest accessability to cover the search space. But the middle point for the whole population is a unique point and cannot be replaced with opposite numbers in the ODE algorithm. For this reason and as an option, a random point between middle point and opposite point can be replaced with opposite number. Figure B.1 and Figure B.2 show the interval and region which are used to generate these points in one-dimensional and two-dimensional spaces, respectively.

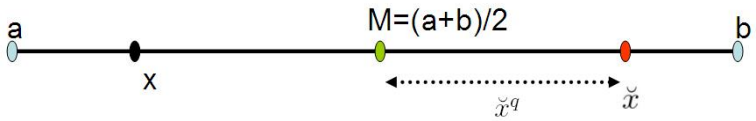


Fig. B.1: Illustration of x and its opposite $\breve{x}$. The quasi-opposite point, $\breve{x}^q$, is generated in the interval $[M, \breve{x}]$.

B.1.1 Quasi-opposition theorem

Mathematically, we can prove that for a black-box optimization problem, the quasi-opposite point $\breve{x}^q$ has a higher chance than opposite point $\breve{x}$ to be closer to the solution. The theorem can be formulated as follows:

Theorem B.1 (Quasi-Opposition Theorem) *Given a guess x, its opposite $\breve{x}$ and quasi-opposite $\breve{x}^q$, and given the probability function $P_r(\cdot)$, we have*

$$P_r\left(|\breve{x}^q - x_s| < |\breve{x} - x_s|\right) > 1/2, \tag{B.1}$$

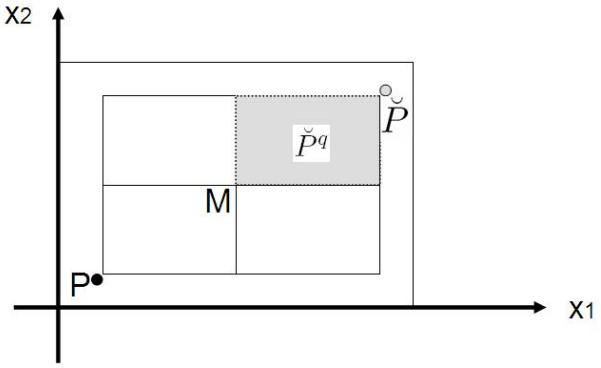


Fig. B.2: For a two-dimensional space, the point P and its opposite $\breve{P}$. The quasi-opposite point, $\breve{P}^q$, is generated in that gray area.

where x_s is the solution for a black-box optimization problem.

Proof Assume that the solution x_s is in one of these intervals: $[a, x]$, $[x, M]$, $[M, \breve{x}]$, $[\breve{x}, b]$ ($[a, x] \cup [x, M] \cup [M, \breve{x}] \cup [\breve{x}, b] = [a, b]$). We investigate all cases:

- $x_s \in [a, x] \vee x_s \in [\breve{x}, b]$ - According to the definition of opposite point, intervals $[a, x]$ and $[\breve{x}, b]$ have the same length, so the probability that the solution is in interval $[a, x]$ or $[\breve{x}, b]$ is equal ($\frac{x-a}{b-a} = \frac{b-\breve{x}}{b-a}$). Now, if the solution is in interval $[a, x]$, definitely, it is closer to $\breve{x}^q$ and in the same manner if it is in interval $[\breve{x}, b]$ it would be closer to $\breve{x}$. So, until now, $\breve{x}^q$ and $\breve{x}$ have the equal chance to be closer to the solution.
- $x_s \in [M, \breve{x}]$ - For this case $\breve{x}^q$ and $\breve{x}$ have the equal chance to be closer to the solution.
- $x_s \in [x, M]$ - For this case, obviously, $\breve{x}^q$ is closer to the solution than $\breve{x}$.

Now, we can conclude that, in overall, $\breve{x}^q$ has a higher chance to be closer to the solution than $\breve{x}$, because for the first two cases they had equal chance and just for last case ($[x, M]$) $\breve{x}^q$ has a higher chance to be closer to the solution.

This proof is for a one-dimensional space, but it can be extended to D-dimensions (similar to Central Opposition Theorem). As proved in one dimensional space, quasi-opposite points have a higher chance to be closer to the solution than opposite points and that is true for all individual dimensions in a D-dimensional space; hence the quasi-opposite points have a higher chance in D-dimensional space to be closer to the solution than opposite points. Now the quasi-oppositional optimization can be defined. That is very similar to the OBO (see section 3.3, page 30).

Definition B.1 (Quasi-Oppositional Optimization) *Let $P(x_1, x_2, ..., x_D)$ be a point in a D-dimensional space (i.e. a candidate solution) and $\breve{P}^q(\breve{x}_1^q, \breve{x}_2^q, ..., \breve{x}_D^q)$ be a quasi-opposite point (see figure B.2). Assume $f(\cdot)$ is a fitness function which is used to measure the candidate's fitness. Now, if $f(\breve{P}^q) \geq f(P)$, then point P can be replaced with $\breve{P}^q$; otherwise we continue with P. Hence, we continue with the fitter one.*

As mentioned before, QODE and ODE have the same population initialization and generation jumping schemes, the only difference is that the opposite points are replaced with quasi-opposite points. Algorithm 6 presents all details for the QODE.

B.1.2 Experimental validation

A set of 15 well-known scalable benchmark functions (chosen from the main test suite), which contains 7 unimodal and 8 multimodal functions, has been selected for performance verification of QODE. Furthermore, test functions with two different dimensions (D and 2D) have been employed in the conducted experiments. By this way, the classical differential evolution (DE), opposition-based DE (ODE), and quasi-oppositional DE (QODE) are compared on 30 minimization exercises. The 13 functions (out of 15) have an optimum in the center of search space. To make it asymmetric, the search space for all of these functions is shifted as follows:

Algorithm 6 Quasi-Oppositional Differential Evolution (QODE)

Generate uniformly distributed random population P_0
//Quasi-oppositional initialization
for $i = 0$ to N_p **do**
  for $j = 0$ to D **do**
    $OP_{0i,j} \leftarrow a_j + b_j - P_{0i,j}$ //Calculating opposite point
    $M_{i,j} \leftarrow (a_j + b_j)/2$ //Calculating middle point
    if $(P_{0i,j} < M_{i,j})$ **then**
      $QOP_{0i,j} \leftarrow M_{i,j} + (OP_{0i,j} - M_{i,j}) \times rand(0,1)$ //Calculating quasi-opposite point
    else
      $QOP_{0i,j} \leftarrow OP_{0i,j} + (M_{i,j} - OP_{0i,j}) \times rand(0,1)$ //Calculating quasi-opposite point
    end if
  end for
end for
Select n fittest individuals from set the $\{P_0, QOP_0\}$ as initial population P_0
//DE's regular steps
while (BFV > VTR **and** NFC < $\mathrm{MAX_{NFC}}$) **do**
  for $i = 0$ to N_p **do**
    Select three parents P_{i_1}, P_{i_2}, and P_{i_3} randomly from current population where $i \neq i_1 \neq i_2 \neq i_3$
    $V_i \leftarrow P_{i_1} + F \times (P_{i_2} - P_{i_3})$
    for $j = 0$ to D **do**
      if $(rand(0,1) < C_r)$ **then**
        $U_{i,j} \leftarrow V_{i,j}$
      else
        $U_{i,j} \leftarrow P_{i,j}$
      end if
    end for
    Evaluate U_i
    if $(f(U_i) \leq f(P_i))$ **then**
      $P'_i \leftarrow U_i$
    else
      $P'_i \leftarrow P_i$
    end if
  end for
  $P \leftarrow P'$
  //Quasi-oppositional generation jumping
  if $(rand(0,1) < J_r)$ **then**
    for $i = 0$ to N_p **do**
      for $j = 0$ to D **do**
        $OP_{i,j} \leftarrow \mathrm{MIN}_j^p + \mathrm{MAX}_j^p - P_{i,j}$ //Calculating opposite point
        $M_{i,j} \leftarrow (\mathrm{MIN}_j^p + \mathrm{MAX}_j^p)/2$ //Calculating middle point
        if $(P_{i,j} < M_{i,j})$ **then**
          $QOP_{i,j} \leftarrow M_{i,j} + (OP_{i,j} - M_{i,j}) \times rand(0,1)$ //Calculating quasi-opposite point
        else
          $QOP_{i,j} \leftarrow OP_{i,j} + (M_{i,j} - OP_{i,j}) \times rand(0,1)$ //Calculating quasi-opposite point
        end if
      end for
    end for
    Select n fittest individuals from set the $\{P, QOP\}$ as current population P
  end if
end while

If O.P.B.: $-a \leq x_i \leq a$ and $f_{min} = f(0, ..., 0) = 0$
then S.P.B.:$-a + \frac{a}{2} \leq x_i \leq a + \frac{a}{2}$,

where O.P.B. and S.P.B. stand for *Original Parameter Bounds* and *Shifted Parameter Bounds*, respectively.

Setting control parameters - Parameter settings for all conducted experiments in this section are as before ($N_p = 100$, $F = 0.5$, $C_r = 0.9$, $J_{r_{ODE}} = 0.3$, $\text{MAX}_{\text{NFC}} = 10^6$, and VTR= 10^{-8}). Just a new jumping rate for QODE, $J_{r_{QODE}}$, is set to 0.05. It is set to a smaller value ($J_{r_{QODE}} = \frac{1}{6} J_{r_{ODE}}$), because the trials showed that the higher jumping rates can rapidly reduce the diversity of the population and cause a premature convergence. This was predictable for QODE, because instead of an opposite point, a random point between middle point and the opposite point is utilized and hence the variable's search interval is prone to be shrunk very fast. A complementary study is required to determine an optimal value/interval for QODE's jumping rate.

Results - Results of applying DE, ODE, and QODE to solve 30 test problems are given in Table B.1. As seen, QODE outperforms others (DE and ODE) on 22 functions, while ODE surpasses DE and QODE on 6 functions and DE can just outperform ODE and QODE on one function. DE performs slightly better than ODE and QODE in terms of average success rate (0.90, 0.88, and 0.86, respectively).

B.2 ODE with Variable Jumping Rate (ODEVJR)

In this section, a time varying jumping rate (TVJR) model for opposition-based differential evolution (ODE) has been investigated. According to this model, the jumping rate changes during the evolution based on the number of function evaluations. The same test suite

Table B.1: Comparison of DE, ODE, and QODE. D: Dimension, NFC: Number of function calls (average over 50 trials), SR: Success rate, SP: Success performance.

		DE			*ODE*			*QODE*		
F	*D*	*NFC*	*SR*	*SP*	*NFC*	*SR*	*SP*	*NFC*	*SR*	*SP*
f_1	30	86072	1	86072	50844	1	50844	42896	1	**42896**
	60	154864	1	154864	101832	1	101832	94016	1	**94016**
f_2	30	95080	1	95080	56944	1	56944	47072	1	**47072**
	60	176344	1	176344	117756	1	117756	105992	1	**105992**
f_3	20	174580	1	174580	177300	1	177300	116192	1	**116192**
	40	816092	1	816092	834668	1	834668	539608	1	**539608**
f_5	10	323770	0.96	337260	75278	0.92	**81823**	181100	1	181100
	20	811370	0.08	10142125	421300	0.16	**2633125**	615280	0.16	3845500
f_6	30	111440	0.96	116083	74717	0.92	**81214**	100540	0.80	125675
	60	193960	1	193960	128340	0.68	188735	115280	0.68	**169529**
f_7	30	18760	1	18760	10152	1	10152	9452	1	**9452**
	60	33128	1	33128	11452	1	**11452**	14667	0.84	17461
f_8	30	168372	1	168372	100280	1	100280	82448	1	**82448**
	60	294500	1	294500	202010	0.96	**210427**	221850	0.72	308125
f_{15}	30	101460	1	101460	70408	1	70408	50576	1	**50576**
	60	180260	0.84	215000	121750	0.60	**202900**	98300	0.40	245800
f_{18}	10	191340	0.76	**252000**	213330	0.56	380900	247640	0.48	515900
	20	288300	0.35	824000	253910	0.55	461700	193330	0.68	**284300**
f_{19}	30	385192	1	385192	369104	1	369104	239832	1	**239832**
	60	–	0	–	–	0	–	–	0	–
f_{21}	30	183408	1	183408	167580	1	167580	108852	1	**108852**
	60	318112	1	318112	274716	1	274716	183132	1	**183132**
f_{23}	30	40240	1	40240	26400	1	26400	21076	1	**21076**
	60	73616	1	73616	64780	1	64780	64205	1	**64205**
f_{31}	30	386920	1	386920	361884	1	361884	291448	1	**291448**
	60	432516	1	432516	425700	0.96	443438	295084	1	**295084**
f_{41}	10	19324	1	19324	16112	1	16112	13972	1	**13972**
	20	45788	1	45788	31720	1	31720	23776	1	**23776**
f_{56}	10	37260	1	37260	26108	1	26108	18944	1	**18944**
	20	176872	1	176872	57888	1	57888	40312	1	**40312**
	SR_{ave}		**0.90**			0.88			0.86	

(used for QODE) has been employed to compare performance of DE and ODE with variable jumping rate settings.

Generally speaking, parameter control in evolutionary algorithms (EAs) can be performed in following three ways [Eiben and Hinterding 1999]: Deterministic, adaptive, and self-adaptive. The first one uses a predefined rule to modify the parameter value without gaining any feedback from the evolution process. The second one changes the parameter value based on the information which receives from the search process. The third one

utilizes the same evolutionary approach not only to solve the problem but also to optimize own control parameters by encoding some strategic parameters inside the population [Rudolph 2001; Hildebrand et al. 1999].

The proposed idea in this section is similar to Das *et al.* work [2005b]. They utilized time varying approach for setting of the scale factor F in differential evolution (DE), which can be considered as a deterministic approach according to the mentioned categorization.

B.2.1 Investigated jumping rate models

For opposition-based differential evolution (ODE), a constant value for jumping rate was used. Here, two types of varying jumping rates are investigated (linearly increasing and decreasing functions). Three proposed settings for J_r are as follows:

- J_r (constant)= $J_{r_{ave}}$,
- $J_r(\text{TVJR1}) = (J_{r_{max}} - J_{r_{min}}) \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$,
- $J_r(\text{TVJR2}) = (J_{r_{max}} - J_{r_{min}}) - (J_{r_{max}} - J_{r_{min}}) \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$,

where $J_{r_{ave}}$, $J_{r_{max}}$, and $J_{r_{min}}$ are the average, maximum, and minimum jumping rates, respectively. MAX_{NFC} and NFC are the maximum number of function calls and the current number of function calls, respectively.

In order to support a fair comparison between these three different jumping rate settings, the average jumping rate should be the same for all of them. Obviously, we should have $J_{r_{ave}} = \frac{(J_{r_{max}} + J_{r_{min}})}{2}$. Following values for these parameters are selected: $J_{r_{ave}} = 0.3$ and $J_{r_{min}} = 0$ (no jumping), so $J_{r_{max}} = 0.6$. Figure B.3 shows the corresponding diagrams (jumping rate vs. NFCs) for three following settings:

- $J_r(\text{constant}) = 0.3$,

- $J_r(\text{TVJR1}) = 0.6 \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$,
- $J_r(\text{TVJR2}) = 0.6 - 0.6 \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$.

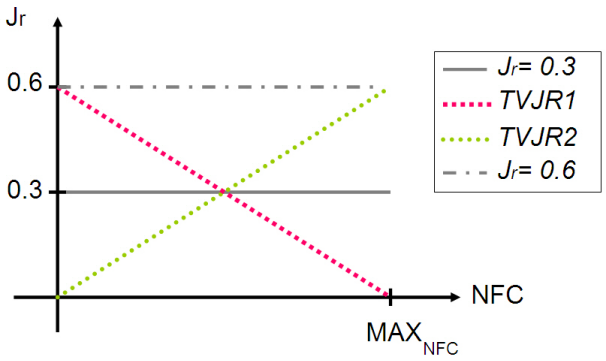


Fig. B.3: Jumping rate vs. NFCs for $J_r(\text{ODE}) = 0.3$, $J_r(\text{TVJR1}) = 0.6 \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$, and $J_r(\text{TVJR2}) = 0.6 - 0.6 \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$.

J_r(TVJR1) represents higher jumping rate during exploration and lower jumping rate during exploitation (fine-tuning); J_r(TVJR2) performs exactly in reverse manner. By these settings, we can investigate effects of generation jumping during optimization process.

B.2.2 Empirical results

The test set and all parameter settings are the same as for QODE. The only difference is the maximum number of function calls, which is 2×10^5 for $f_1, f_2, f_3, f_6, f_8, f_{15}, f_{21}$; 5×10^5 for $f_5, f_{18}, f_{19}, f_{31}$; and 5×10^4 for $f_7, f_{23}, f_{41}, f_{56}$. Results of applying DE, ODE ($J_r = 0.3$), ODE (TVJR1), and ODE (TVJR2) to solve 15 test problems are given in Table B.2. As seen, ODE (TVJR1) delivers best success performance (SP) for 13 benchmark functions, while this number for DE, ODE ($J_r = 0.3$), and ODE (TVJR2) is 0, 1, and 1, respectively.

Table B.2: Comparison of DE, ODE ($J_r = 0.3$), ODE (TVJR1), and ODE (TVJR2). D: Dimension, NFC: Number of function calls (average over 50 trials), SR: Success rate, SP: Success performance.

		DE			*ODE ($J_r = 0.3$)*			*ODE (TVJR1)*			*ODE (TVJR2)*		
F	D	*NFC*	*SR*	*SP*	*NFC*	*SR*	*SP*	*NFC*	*SR*	*SP*	*NFC*	*SR*	*SP*
f_1	30	87748	1	87748	47716	1	47716	42300	1	**42300**	66305	1	66305
f_2	30	96488	1	96488	53304	1	53304	45720	1	**45720**	72990	1	72990
f_3	20	177880	1	177880	168680	1	168680	159775	1	**159775**	175460	1	175460
f_5	10	328844	1	328844	65056	0.64	101650	59063	0.80	**73829**	136070	1	136070
f_6	30	113428	1	113428	64920	0.75	86560	63594	0.90	**70660**	86235	1	86235
f_7	30	25140	1	25140	8328	1	8328	6080	1	**6080**	14175	1	14175
f_8	30	169152	1	169152	98296	1	98296	88355	1	**88355**	117095	1	117095
f_{15}	30	101460	1	101460	70408	1	70408	65247	0.95	**68681**	82245	1	82245
f_{18}	10	215260	0.56	384393	168470	0.76	**221671**	188440	0.65	289908	379660	0.60	632767
f_{19}	30	385192	1	385192	369104	1	369104	389955	1	389955	360595	1	**360595**
f_{21}	30	187300	1	187300	155636	1	155636	146795	1	**146795**	167685	1	167685
f_{23}	30	41588	1	41588	23124	1	23124	20290	1	**20290**	29165	1	29165
f_{31}	30	411164	1	411164	337532	1	337532	326350	1	**326350**	377425	1	377425
f_{41}	10	19528	1	19528	15704	1	15704	14270	1	**14270**	17735	1	17735
f_{56}	10	37824	1	37824	24260	1	24260	21400	1	**21400**	28710	1	28710
	SR_{ave}	0.97			0.94			0.95			0.97		

Table B.3: Pairwise comparison of DE, ODE ($J_r = 0.3$), ODE (TVJR1), and ODE (TVJR2). Given number in each cell shows for how many functions the algorithm in each row outperforms the corresponding algorithm in each column. The last column shows the total numbers (number of cases which the algorithm outperforms other competitors).

	DE	*ODE ($J_r = 0.3$)*	*ODE (TVJR1)*	*ODE (TVJR2)*	Total
DE	-	0	1	1	**2**
ODE ($J_r = 0.3$)	15	-	2	12	**29**
ODE (TVJR1)	14	13	-	14	**41**
ODE (TVJR2)	14	3	1	-	**18**

Pairwise comparison of these algorithms is presented in Table B.3. The given number in each cell indicates for how many functions the algorithm in each row outperforms the corresponding algorithm in each column. The last column of the table shows the total numbers (number of cases which the algorithm outperforms other competitors); by comparing these numbers, the following ranking is obtained: ODE (TVJR1) (best), ODE ($J_r = 0.3$), ODE (TVJR2), and DE.

The average success rate (shown in the last row of the Table B.2) for DE and ODE (TVJR2) is marginally better than the other two competitors.

B.3 Summary

In this Appendix, the quasi-oppositional DE (QODE), an enhanced version of the opposition-based differential evolution (ODE), was introduced. Both algorithms (ODE and QODE) use the same schemes for population initialization and generation jumping. But, QODE uses quasi-opposite points instead of opposite points. The presented mathematical proof confirms that these points have a higher chance than opposite points to be closer to the solution. Experimental results, conducted on 30 - eccentric minimum - test problems, clearly show that QODE outperforms ODE.

As another enhancement direction, the time varying jumping rate for opposition-based differential evolution was proposed and two behaviorally reverse versions (linearly decreasing and increasing functions) were compared with the constant setting. The results show that the linearly decreasing jumping rate performs better than constant setting and also than linearly increasing policy. This means that generation jumping in the exploration time is more desirable than during exploitation. Because we are faced with shrunken search space during the fine-tuning, and the jumping of the individuals may not be advantageous (because the point and the opposite-point are very close together). There is no exact border between exploration and exploitation stage. Hence, the gradual behavior for the decreasing and increasing functions are proposed.

The proposed jumping rate function utilizes the maximum number of function calls ($\mathrm{MAX_{NFC}}$) which may not be exactly known for the black-box optimization problems; this can be regarded as a disadvantage for this method. Adaptive setting of the jumping rate can be a desirable solution.

Bibliography

Ali, M. and Törn, A. 2000. Optimization of carbon and silicon cluster geometry for tersoff potential using differential evolution. In *Computational Chemistry and Molecular Biology : Local and Global Approaches, C.A. Floudas and P.M. Pardalos (Eds.), Kluwer Academic Publisher*. 287–300.

Ali, M. and Törn, A. 2004. Population set-based global optimization algorithms: Some modifications and numerical studies. *Journal of Computers and Operations Research 31(10)*, 1703–1725.

Aluffi-Pentini, F., Parisi, V., and Zirilli, F. 1985. Global optimization and stochastic differential equations. *Journal of Optimization Theory and Applications 47*, 1–16.

Angeline, P. J. 1998. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In *Proceedings of the 7th International Conference on Evolutionary Programming VII*. Springer-Verlag, London, UK, 601–610.

Babu, B. and Sastry, K. 1999. Estimation of heat transfer parameters in a tricklebed reactor using differential evolution and orthogonal collocation. *Computers and Chemical Engineering 23*, 327–339.

Bäck, T. 1996. *Evolutionary Algorithms in Theory and Practice : Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press Inc., USA.

Bäck, T., Hammel, U., and Schwefel, H.-P. 1997. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation 1(1)*, 3–17.

Benke, K. and Skinner, D. 1991. A direct search algorithm for global optimization of multivariate functions. *The Australian Computer Journal 23*, 73–85.

Bergh, F. V. D. and Englebrecht, A. 2004. A cooperative approach to particle swarm optimization. *Journal of IEEE Transactions on Evolutionary Computation 8(3)*, 225–239.

Bohachevsky, M., Johnson, M., and Stein, M. 1986. Generalized simulated annealing for function optimization. *Technometrics 28*, 209–217.

Bohuslav, R. and Michal, K. 2001. Differential evolution algorithm in the earthquake hypocenter location. *Pure and Applied Geophysics 158*, 667–693.

Breiman, L. and Cutler, A. 1993. A deterministic algorithm for global optimization. *Mathematical Programming 58*, 179–199.

Brest, J., Greiner, S., Bošković, B., Mernik, M., and Žumer, V. 2006. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *Journal of IEEE Transactions on Evolutionary Computation 10(6)*, 646–657.

Chakraborti, N., De, P., and Prasad, R. 2002. Genetic algorithms based structure calculations for hydrogenated silicon clusters. *Materials letters 55(1)*, 20–26.

Chellapilla, K. 1998. Combining mutations operators in evolutionary programming. *Journal of IEEE Transactions on Evolutionary Computation 2*, 91–96.

Cheng, H., Jiang, X., Sun, Y., and Wang, J. 2001. Color image segmentation: advances and prospects. *Journal of Pattern Recognition 34*, 2259–2281.

Crutchley, D. and Zwolinski, M. 2004. Globally convergent algorithms for DC operating point analysis for nonlinear circuits. *Journal of IEEE Transactions on Evolutionary Computation 7(1)*, 2–10.

Das, S., Konar, A., and Chakraborty, U. 2005a. Improved differential evolution algorithms for handling noisy optimization problems. In *Proceedings of IEEE Congress on Evolutionary Computation Conference*. Napier University, Edinburgh, UK, 1691–1698.

Das, S., Konar, A., and Chakraborty, U. 2005b. Two improved differential evolution schemes for faster global search. In *Proceedings of the 2005 conference on Genetic and evolutionary computation (GECCO-2005)*. Washington DC, USA, 991–998.

Dekkers, A. and Aarts, E. 1991. Global optimization and simulated annealing. *Mathematical Programming 50*, 367–393.

Dixon, L. and Szegö, G. 1975. *Towards Global Optimization*. North Holland, New York.

Dixon, L. and Szegö, G. 1978. *Towards Global Optimization*. Vol. 2. North Holland, New York.

Dorigo, M. and Stützle, T. 2004. *Ant Colony Optimization*. MIT Press, USA.

Eiben, A. and Hinterding, R. 1999. Paramater control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation 3(2)*, 124–141.

Fan, H.-Y. and Lampinen, J. 2003. A trigonometric mutation operation to differential evolution. *Global Optimization 27(1)*, 105–129.

Fathi, M. and Hildebrand, L. 1997. Model-free optimization of fuzzy rule-based systems using evolution strategies. *IEEE Transactions on Systems, Man, and Cybernetics, Part B 27(2)*, 270–277.

Feoktistov, V. 2006. *Differential Evolution: In Search of Solutions*. Springer, USA.

Fischer, M., Reismann, M., and Hlavackova-Schindler, K. 1999. Parameter estimation in neural spatial interaction modelling by a derivative free global optimization method. In *Proceedings of the IV Intnational Conference on geocomputation*. Mary Washington College, Fredericksburg, VA, USA.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Han, K.-H. and Kim, J.-H. 2004. Quantume-inspired evolutionary algorithms with a new termination criterion, H gate, and two phase scheme. *Journal of IEEE Transactions on Evolutionary Computation 8(2)*, 156–169.

Hildebrand, L., Reusch, B., and Fathi-Torbaghan, M. 1999. Directed mutation: A new self-adaptation for evolution strategies. In *Proceedings of the Congress on Evolutionary Computation (CEC-1999), IEEE Publications*. Vol. 2. Washington, DC, USA, 1550–1557.

Himmelblau, D. 1972. *Applied Nonlinear Programming*. McGraw-Hill, New York.

Hrstka, O. and Kučerová, A. 2004. Improvement of real coded genetic algorithm based on differential operators preventing premature convergence. *Journal of Advance in Engineering Software 35*, 237–246.

Jonasson, E. and Rahnamayan, S. 2006. Differential evolution with fittest individual local tuning. In *Proceedings of 10th World Multi Conference on Systemics, Cybernetics and Informatics (WMSCI-2006)*. Orlando, USA.

Joshi, R. and Sanderson, A. 1999. Minimal representation multisensor fusion using differential evolution. *Journal of IEEE Transactions on Evolutionary Computation 29(1)*, 63–76.

Kachouie, N., Fieguth, P., and Rahnamayan, S. 2006. An elliptical level set method for automatic TRUS prostate image segmentation. In *Proceedings of 6th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT-2006)*. Vancouver, Canada, 191–196.

Kaelo, P. and Ali, M. M. 2006. Probabilistic adaptation of point generation schemes in some global optimization algorithms. *Optimization Methods and Software 27(3)*, 343–357.

Koumousis, V. and Katsaras, C. 2006. A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *Journal of IEEE Transactions on Evolutionary Computation 10(1)*, 19–28.

Krink, T., Filipie, B., and Fogel, G. 2004. Noisy optimization problems - a particular challenge for differential evolution? In *Proceedings of the 2004 IEEE World Congress on Computational Intelligence (CEC-2004)*. Piscataway, NJ, USA, 332–339.

Kukkonen, S. and Lampinen, J. 2004. An extension of generalized differential evolution for multi-objective optimization with constraints. In *Proceedings of Parallel Problem Solving from Nature - PPSN VIII, Springer-Verlag LNCS*. Vol. 3242. 752–761.

Lampinen, J. 2004. A constraint handling approach for the differential evolution. In *Proceedings of the 2002 IEEE World Congress on Computational Intelligence (CEC-2002)*. Vol. 2. Birmingham, UK, 752–761.

Lampinen, J. and Zelinka, I. 1999a. *Mechanical Engineering Design Optimization by Differential Evolution*. McGraw-Hill, London (UK).

Lampinen, J. and Zelinka, I. 1999b. Mixed variable non-linear optimization by differential evolution. In *Proceedings of Nostradamus-99, 2nd International Prediction Conference*. Technical University of Brno, Zlin, Czech Republic, 45–55.

Lee, C. Y. and Yao, X. 2004. Evolutionary programming using mutations based on the lévy probability distribution. *Journal of IEEE Transactions on Evolutionary Computation 8(1)*, 1–13.

Leung, Y.-W. and Wang, Y. 2001. An orthogonal genetic algorithm with quantization for global numerical optimization. *Journal of IEEE Transactions on Evolutionary Computation 5(1)*, 41–53.

Levy, A. and Montalvo, A. 1985. The tunneling algorithm for the global minimization of functions. *Society for Industrial and Applied Mathematics 6*, 15–29.

Liu, J. and Lampinen, J. 2005. A fuzzy adaptive differential evolution algorithm. *Journal of Soft Computing-A Fusion of Foundations, Methodologies and Applications 9(6)*, 448–462.

Martens, J. and Meesters, L. 1998. Image dissimilarity. *Signal Processing 70*, 1164–1175.

McCormick, G. 1982. *Applied Nonlinear Programming, Theory, Algorithms and Applications*. John Wiley and Sons, New York.

Mezura-Montes, E., Velázquez-Reyes, J., and Coello, C. A. C. 2006. A comparative study of differential evolution variants for global optimization. In *Proceedings of the 2006 conference on Genetic and evolutionary computation (GECCO-2006)*. Seattle, Washington, USA, 485–492.

Michalewicz, Z. 1996. *Genetic Algorithms+Data Structures=Evolution Programs*. Springer-Verlag, Berlin/Heidelberg/New York.

Moré, J., Garbow, B., and Hillstrom, K. 1981. Testing unconstrained optimization software. *ACM Transaction on Mathematical Software 7*, 17–41.

Noman, N. and Iba, H. 2005. Enhancing differential evolution performance with local search for high dimensional function optimization. In *Proceedings of the 2005 conference on Genetic and evolutionary computation (GECCO-2005)*. Washington DC, USA, 967–974.

Onwubolu, G. C. and Babu, B. 2004. *New Optimization Techniques in Engineering*. Springer, Berlin, New York.

Paterlini, S. and Krink, T. 2004. High performance clustering with differential evolution. In *Proceedings of the 2004 IEEE World Congress on Computational Intelligence (CEC-2004)*. Piscataway, NJ, USA, 2004–2011.

Penev, K. and Littlefair, G. 2005. Free search - a comparative analysis. *Information Sciences 172(1-2)*, 173–193.

Plagianakos, V., Magoulas, G., Nousis, N., and Vrahatis, M. 2001. Training multilayer networks with discrete activation functions. In *Proceedings of INNS-IEEE Int. joint Conf. on neural networks*. Vol. 4. Washington DC, USA, 2805–2810.

Price, K. 1994. Genetic annealing. *Dr. Dobb's Journal 220*, 127–132.

Price, K. 1999. *An Introduction to Differential Evolution*. McGraw-Hill, London (UK).

Price, K. and Storn, R. 1997. Differential evolution: Numerical optimization made easy. *Dr. Dobb's Journal 220*, 18–24.

Price, K., Storn, R., and Lampinen, J. 2005. *Differential Evolution : A Practical Approach to Global Optimization*, 1st ed. Springer-Verlag, Berlin/Heidelberg/Germany.

Price, W. 1977. Global optimization by controlled random search. *Computer Journal 20*, 367–370.

Rahnamayan, S. 2006. A new initialization scheme for evolutionary optimization methods. In *Proceedings of 10th World Multi Conference on Systemics, Cybernetics and Informatics (WMSCI-2006)*. Orlando, USA.

Rahnamayan, S. and Dieras, P. 2007. Efficiency competition on N-queen problem: DE vs. CMA-ES. *Submitted to the IEEE Congress on Evolutionary Computation (CEC-2007), Singapore*.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2005a. Automated snake initialization for the segmentation of the prostate in ultrasound images. In *Proceedings of International Conference on Image Analysis and Recognition (ICIAR-2005), Springer Lecture Notes in Computer Science Series*. Toronto, Canada, 930–937.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2005b. Learning image filtering from a gold sample based on genetic optimization of morphological processing. In *Proceedings of 7th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA-2005), SpringerComputerScience*. Coimbra, Protugal, 478–481.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2005d. Optimization of object extraction based on one user-prepared sample. *Presented at 5th Annual MOPTA Conference, Modeling and Optimization: Theory and Applications, University of Windsor, Windsor, Canada*.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2005e. Recognition of subjective objects based on one gold sample. In *Proceedings of 5th WSEAS International Conference on Signal, Speech and Image Processing*. Corfu, Greece, 309–314.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2005f. Robust object segmentation using genetic optimization of morphological processing chains. In *Proceedings of 5th WSEAS International Conference on Signal, Speech and Image Processing*. Corfu, Greece, 248–253.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2006a. Image thresholding using differential evolution. In *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV-2006)*. Las Vegas, USA, 244–249.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2006b. A novel population initialization method for accelerating evolutionary algorithms. *Elsvier Journal on Computers and Mathematics with Applications (in press)*.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2006c. Opposite of random number instead of second random number. *Presented at 6th Annual MOPTA Conference, Modeling and Optimization: Theory and Applications, University of Waterloo, Waterloo, Canada*.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2006d. Opposition-based differential evolution algorithms. In *Proceedings of the IEEE World Congress on Computational Intelligence (CEC-2006)*. Vancouver, BC, Canada, 7363–7370.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2006e. Opposition-based differential evolution for optimization of noisy problems. In *Proceedings of the IEEE World Congress on Computational Intelligence (CEC-2006)*. Vancouver, BC, Canada, 6756–6763.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2006g. Opposition-based evolutionary algorithms. *Presented at 6th Annual MOPTA Conference, Modeling and Optimization: Theory and Applications, University of Waterloo, Waterloo, Canada*.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2006h. Weighted voting-based robust image thresholding. In *Proceedings of 13th IEEE International Conference on Image Processing (ICIP-2006)*. Atlanta, GA, USA, 1129–1132.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2007a. Opposition-based differential evolution (ODE) with variable jumping rate. In *Proceedings of IEEE Symposium on Foundations of Computational Intelligence (FOCI-2007)*. Honolulu, Hawaii, USA, 81–88.

Rahnamayan, S., Tizhoosh, H., and Salama, M. 2007c. Quasi-oppositional differential evolution (QODE). *Submitted to the IEEE Congress on Evolutionary Computation (CEC-2007), Singapore*.

Rahnamayan, S., Tizhoosh, H., and Salama, M. Dec. 2006f. Opposition-based differential evolution (ODE). *Journal of IEEE Transactions on Evolutionary Computation (in press)*.

Rahnamayan, S., Tizhoosh, H., and Salama, M. March 2007b. Opposition versus randomness in soft computing techniques. *Revised version submitted to the Elsevier Journal on Applied Soft Computing*.

Rahnamayan, S., Tizhoosh, H., and Salama, M. Oct. 2005g. Towards incomplete object recognition. *Journal of World Scientific and Engineering Academy and Society, Transactions on Systems 4(10)*, 1725–1732.

Rahnamayan, S., Tizhoosh, H., and Salama, M. Sep. 2005c. Learning robust object segmentation from user-prepared samples. *Journal of WSEAS Transactions on Computers 4(9)*, 1163–1170.

ROGALSKY, T., DERKSEN, R., AND KOCABIYIC, S. 1999. Differential evolution in aerodynamic optimization. In *Proceedings of the 46th Annual Conference of the Canadian Aeronautics and Space Institute*. Montreal, Canada, 29–36.

RUDOLPH, G. 2001. Self-adaptive mutations may lead to premature convergence. *IEEE Transactions on Evolutionary Computation 5(4)*, 410–414.

SALOMON, M. 2001. Etude de la parallelisation de methodes heuristiques d'optimisation combinatoire. application au recalage d'images medicales. Ph.D. thesis, Universite Louis Pasteur, Strasbourg, France.

SALOMON, R. 1995. Reevaluating genetic algorithms performance under coordinate rotation of benchmark functions. *BioSystems 39(3)*, 263–278.

SCHWEFEL, H.-P. 1995. *Evolution and Optimization Seeking*. John Wiley & Sons, New York.

SCHWEFEL, H.-P. 2003. *Computational Intelligence: Theory and Practice*. Springer-Verlag New York, USA.

SEZAN, M. 1985. A peak detection algorithm and its application to histogram-based image data reduction. *Journal of Computer Vision, Graphics, Image Processing 29*, 47–59.

SEZGIN, M. AND SANKUR, B. 2004. Survey over image thresholding techniques and quantative performance evaluation. *Journal of Electronic Imaging 13(1)*, 146–165.

SHI, Y.-J., TENG, H.-F., AND LI, Z.-Q. 2005. Cooperative co-evolutionary differential evolution for function optimization. In *Proceedings of First International Conference in Advances in Natural Computation (ICNC-2005)*. Changsha, China, 1080–1088.

Storn, R. 1996. On the usage of differential evolution for function optimization. In *Proceedings of Biennial conference of the North American fuzzy information processing society*. Berkeley, California, USA, 519–523.

Storn, R. and Price, K. 1997a. Differential evolution- a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization, Kluwer 11*, 341–359.

Storn, R. and Price, K. 1997b. Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization 11*, 341–359.

Storn, R. and Price, K. March 1995. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Tech. Rep. TR-95-012, ICSI.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. P., Auger, A., and Tiwari, S. May 2005. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. Tech. Rep. 2005005, Kanpur Genetic Algorithms Laboratory, IIT Kanpur, Nanyang Technological University, Singapore And KanGAL.

Sun, J., Zhang, Q., and Tsang, E. P. 2005. DE/EDA: A new evolutionary algorithm for global optimization. *Journal of Information Sciences 169*, 249–262.

Tasoulis, D., Pavlidis, N., Plagianakos, V., and Vrahatis, M. 2004. Parallel differential evolution. In *Proceedings of the Congress on Evolutionary Computation (CEC-2004), IEEE Publications*. Vol. 2. 2023–2029.

Teo, J. 2006. Exploring dynamic self-adaptive populations in differential evolution. *Soft Computing - A Fusion of Foundations, Methodologies and Applications 10(8)*.

Tizhoosh, H. 2005a. Image thresholding using type II fuzzy sets. *Journal of Pattern Recognition 38*, 2363–2372.

Tizhoosh, H. 2005b. Opposition-based learning: A new scheme for machine intelligence. In *Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA-2005)*. Vienna, Austria, 695–701.

Tizhoosh, H. 2005c. Reinforcement learning based on actions and opposite actions. In *Proceedings of the International Conference on Artificial Intelligence and Machine Learning (AIML-2005)*. Cairo, Egypt.

Tizhoosh, H. 2006. Opposition-based reinforcement learning. *Journal of Advanced Computational Intelligence and Intelligent Informatics 10(3)*, 578–585.

Tsai, J.-T., Liu, T.-K., and Chou, J.-H. 2004. Hybrid taguchi-genetic algorithm for global numerical optimization. *Journal of IEEE Transactions on Evolutionary Computation 8(4)*, 365–377.

Tu, Z. and Lu, Y. 2004. A robust stochastic genetic algorithm for global numerical optimization. *Journal of IEEE Transactions on Evolutionary Computation 8(5)*, 456–470.

Ursem, R. and Vadstrup, P. 2004. Parameter identification of induction motors using differential evolution. *Applied Soft Computing 4(1)*, 49–64.

Vesterstroem, J. and Thomsen, R. 2004. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the Congress on Evolutionary Computation (CEC-2004), IEEE Publications*. Vol. 2. San Diego, California, USA, 1980–1987.

Wang, Z. and Bovik, A. 2002. A universal image quality index. *Journal of IEEE Signal Processing Letters 9(3)*, 81–84.

Winkler, S. 2000. Vision models and quality metrics for image processing applications. Ph.D. thesis, École Polytechnique Fédérale de Lausanne, Switzerland.

Wolfe, M. 1978. *Numerical Methods for Unconstrained Optimization.* Van Nostrand Reinhold Company, New York.

Yao, X. and Liu, Y. 1997. Fast evolution strategies. In *Evolutionary Programming VI*, P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, Eds. Springer, Berlin, 151–161.

Yao, X., Liu, Y., and Lin, G. 1999. Evolutionary programming made faster. *Journal of IEEE Transactions on Evolutionary Computation 3(2)*, 82–102.

Yasnoff, W., Mui, J., and Bacus, W. 1977. Error measures for scene segmentation. *Journal of Pattern Recognition 9*, 217–231.