Chapter 3
The Use of Opposition for Decreasing Function Evaluations in Population-Based Search

Mario Ventresca, Shahryar Rahnamayan, and Hamid Reza Tizhoosh

Abstract. This chapter discusses the application of opposition-based computing to reducing the amount of function calls required to perform optimization by population-based search. We provide motivation and comparison to similar, but different approaches including antithetic variates and quasi-randomness/ low-discrepancy sequences. We employ differential evolution and population-based incremental learning as optimization methods for image thresholding. Our results confirm improvements in required function calls, as well as support the oppositional princples used to attain them.

3.1 Introduction

Global optimization is concerned with discovering an optimal (minimum or maximum) solution to a given problem generally within a large search space. In some instances the search space may be simple (i.e. concave or convex optimization can be used). However, most real world problems are multi-modal and deceptive [5] which often causes traditional optimization algorithms to become trapped at local optima. Many strategies have been developed to overcome this for global optimization including, but not limited to simulated annealing [9], tabu search [4], evolutionary algorithms [7] and swarm intelligence [3].

Some of these methods employ a single solution per iteration methodology whereby only one solution is generated and successively perturbed towards more

Mario Ventresca · Hamid Reza Tizhoosh
Department of Systems Design Engineering, The University of Waterloo, Ontario, Canada
e-mail: mventres@pami.uwaterloo.ca,tizhoosh@uwaterloo.ca

Shahryar Rahnamayan
Faculty of Engineering and Applied Science,
The University of Ontario Institute of Technology, Ontario, Canada
e-mail: shahryar.rahnamayan@uoit.ca

Y. Tenne and C.-K. Goh (Eds.): Computational Intelligence in Optimization, ALO 7, pp. 49–71.
springerlink.com

appropriate solutions (i.e. simulated annealing and tabu search). Single-solution methods inherently require low computational time per iteration, however they often lack sufficient diversity to adequately explore the search space. An alternative is via population-based techniques where many solutions are generated at each iteration and all (or most) are used to determine the search direction (i.e. evolutionary and swarm intelligence algorithms). By considering many solutions per iteration these methods will have a higher diversity, however will require a large amount of computation. In this chapter we focus on population-based optimization.

Real-world problems also present the possible issue of complexity in the evaluation of a solution. That is, determining the quality of a given solution is computationally expensive. Investigation into simpler evaluation metrics is one possible direction, however, it may be found that evaluation is still expensive. Then, reducing the time spent (i.e. number of function evaluations) becomes an important goal.

Opposition-based computing (OBC) is a newly devised concept, having as one of its aims, the improvment of convergence rates of algorithms by defining and simultaneously considering pairs of opposite solutions [24]. This advanced convergence rate is also usually accompanied by a more desirable final result. To date, OBC has shown improvements in reinforcement learning [20, 22, 23], evolutionary algorithms [14, 15, 16], ant colony optimization [12], simulated annealing [28], estimation of distribution [29], and neural networks [26, 27, 30].

In this chapter we discuss the application of OBC to reducing the number of function calls required to achieve a desired accuracy for population-based searches. We show theoretical reasoning behind OBC (which has roots in monotonic optimization) and provide mathematical motivations for its ability to reduce function calls and improve accuracy of simulation results. The key factor to accomplishing this is via simultaneous consideration of negatively associated variables and their affect on the target evaluation function and search process. We choose to highlight the improvements for the task of image thresholding using differential evolution and population-based incremental learning.

The rest of this chapter is organized as follows: Section 3.2 discusses the theoretical motivations behind our approach. Differential evolution and population-based incremental learning are discussed in Section 3.3 as are their respective opposition-based counterparts. The experimental setup is given in Section 3.4 and results are presented in Section 8.4. Conclusions are give in Section 3.6.

3.2 Theoretical Motivations

In this section we introduce notations and definitions required to explain the concept of opposition and its ability to reduce the number of function calls. We also provide a brief comparison of OBC to antithetic variates and quasi-random/low-discrepancy sequences.

3.2.1 Definitions and Notations

In the following definitions, assume that $\mathscr{A} \subseteq \Re^d$ is non-empty, compact and d-dimensional. Without loss of generality, $f\colon \mathscr{A} \to \Re$ is a continuous function to be maximized. We assume all $\mathscr{A}$ are feasible.

The purpose of a global search method is to discover the global optima (either minimum or maximum) of a given function and not converge to one of the local optima.

Definition 1 (Global Optima). A solution $\theta^* \in \mathscr{A}$ is a global optima if $f(\theta) \leq f(\theta^*)$ for all $\theta \in \mathscr{A}$. There may exist more than one global optima.

Definition 2 (Local Optima). A solution $\theta' \in \mathscr{A}$ is a local optima if there exists a ε-neighborhood $N_\varepsilon(\theta')$ with radius $\varepsilon > 0$ where $g(\theta', \theta) < \varepsilon$ for distance function g and $\theta \in \mathscr{A} \cap N_\varepsilon(\theta')$, and $f(\theta) \leq f(\theta')$.

Recent research [1, 25, 32] has shown the benefit of utilizing monotonic transformations of the evaluation criteria as a means of discovering global optima. This causes a reordering of the solutions and a gradient-based method can be used to search the reordered space. An issue with these convexification and concavification methods which transform certain functions to a monotonic form is that the mapping must be known a priori. Otherwise, optimization on the transformed function is unreliable.

Definition 3 (Monotonicity). Function $\phi\colon \Re \to \Re$ is monotonic if for $x, y \in \Re$ and $x < (>) y$, then $\phi(x) \leq (\geq)\phi(y)$. A strictly monotonic function is one which does not permit equality, (i.e. $\phi(x) < (>)\phi(y)$).

Theoretically, a monotonic transformation is ideal, however, OBC does not require it. Instead, opposition extends the monotonic global search idea through the use of opposites solutions, which are simultaneously considered and the more desirable (w.r.t. f and the problem definition) is used during the search.

Definition 4 (Opposite). A pair $(x, \breve{x}) \in \mathscr{A}$ are opposites if there exists a function $\Phi\colon \mathscr{A} \to \mathscr{A}$ where $\Phi(x) = y$ and $\Phi(y) = x$. The breve notation will be used to denote the opposite element (i.e. $\breve{x} = \Phi(x) = y$).

The function Φ referred to in Definition 4 is the key to employing opposition-based techniques. This determines which elements are opposites, and a poorly selected function could lead to poor performance (see the following section).

Definition 5 (Opposite Mapping). A one-to-one function $\Phi\colon \mathscr{A} \to \mathscr{A}$ where every pair $x, \breve{x} \in \mathscr{A}$ are unique (i.e. for $z \in \mathscr{A}$, if $\Phi(x) = y$ and $\Phi(y) = x$ then there does not exist $\Phi(y) = z$ or $\Phi(x) = z$).

Φ can be determined via prior knowledge, intuition or through some a priori or online learning procedure. Simultaneous use of the opposites (for example, a maximization problem) is easily accomplished by allowing $f(\theta) = f(\breve{\theta}) = \max(f(\theta), f(\breve{\theta}))$ and searching for a solution S which corresponds to the most desirable solution.

$$S = \arg\max_{\theta} f(\theta). \tag{3.1}$$

3.2.2 Consequences of Opposition

As with a monotonic transformation the "optimal" opposition map is one which effectively reorders elements of $\mathscr{A}$ such that $\phi(f)$ is monotonic. Under this transformation $cor(X,\breve{X}) \leq 0$ for $X,\breve{X} \subset \mathscr{A}$, as is shown in Figure 3.1.

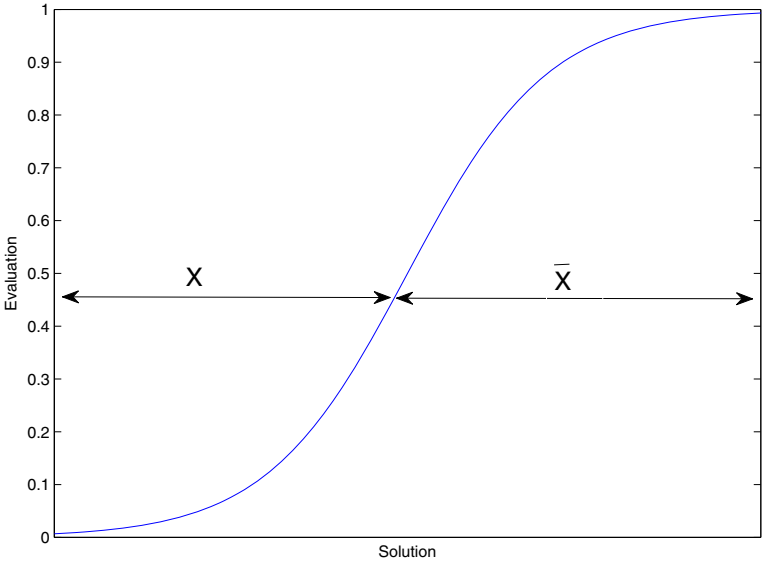

Fig. 3.1 Example of a transformed evaluation funtion to a monotonic function. The values in X and $\breve{X}$ are negatively correlated. The original function (not shown) could have been any nonlinear, continuous function

The implementation of opposition for an optimization problem involves the simultaneous examination of $x,\breve{x}$ and returns the most desirable. That is, we aim for a negative correlation between the two guesses. A consequence of Φ is an effective halving of possible evaluations (as shown in Figure 3.2) where $f(\theta) = f(\breve{\theta}) = \max(f(\theta), f(\breve{\theta}))$. The halving results from full information of the transformed function such that the opposite solution is not required to be observed to compute the max operation. In the more general case, it is sufficient to determine a function such that $Pr(\max(f(\theta_1), f(\breve{\theta}_1) < \max(f(\theta_1), f(\theta_2))) > 0.5$, for $\theta_1, \theta_2 \in \mathscr{A}$.

A further consequence of simultaneous consideration of opposites is a provably more desirable $\mathbb{E}[f]$ and lower variance [28]. Alone this does not guarantee a higher quality outcome; the probability density function of the opposite-transformed function values should also be more desirable than the joint p.d.f. of random sampling (i.e. the distribution corresponding to $Pr(\max(x_1, x_2))$).

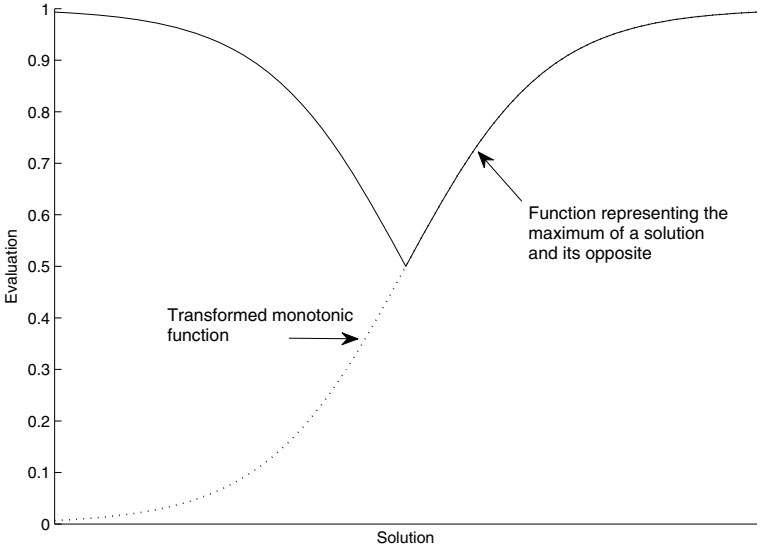

Fig. 3.2 Taking $f(x) = \max(f(x), f(-x))$, we see that the possible evaluations in the search space have been halved in the optimal situation of full knowledge. In the general case, the transformed function will have a more desirable mean and lower variance

While not investigated in this chapter we make the observation that successive applications of different opposite maps will lead to further smoothing of f. For example,

$$f^2(\theta) = \max(f(z = \arg\max_{\theta} f(\theta)), f(\breve{z})) \tag{3.2}$$

where $\breve{z}$ is determined via opposite map $\Phi_2 \neq \Phi_1$ and superscript f^2 indicates the two applications of opposite mappings. In the limit,

$$\lim_{i \to \infty} f^i(\theta) = \max(f^i(z_i = \arg\max_{\theta_i} f^{i-1}(\theta_i)), f(\breve{z}_i)) = f(\theta^*) \tag{3.3}$$

for $i > 0$ and global optima $f(\theta^*)$. Effectively, this flattens the entire error surface of f, except for the global optimal(s). A more feasible alternative is to use $k > 0$ transformations which give reasonable results where $0 \leq |f^{k-1} - f^k| < \varepsilon$ does not diminish greatly.

3.2.3 *Lowering Function Evaluations*

We briefy discuss conditions on the design of the opposition map Φ which often lead to improvements over purely random sampling with respect to lowering function evaluations. If using an algorithm solely based on randomly generating solutions to the problem at hand then we require that for some $\varepsilon > 0$ and $\delta > 0.5$

$$Pr(\max(f(x), f(\check{x})) - \max(f(x), f(y)) > \varepsilon) > \delta \tag{3.4}$$

where $x, y, \check{x} \in \mathscr{A}$. That is, the distribution $\max(x, \check{x})$ must be less desirable than the distribution of i.i.d. random guesses. If this condition is met, then the probability that the optimal solution (or higher quality) is discovered is higher using opposite guesses. The goal in developing Φ is to maximize ε and δ. A similar goal is to determine Φ such that $\mathbb{E}[g(x, \check{x})]$ is maximized for some distance function g. Satisifying this condition implies (3.4).

Thus, probabilistically we expect a lower number of function calls to find a solution of a given thresholded quality. If employing this strategy within a guided search method the dynamics of the algorithm must be considered to assure the guarantee (i.e. the algorithm adds bias to the search which affects the convergence rate).

Practically, the simplest manner to decide a satisfactory Φ is through intuition or prior knowledge about f. A possibility is to utilize modified low-discrepancy sequences (see below), which aim to distribute guesses evenly throughout the search space.

3.2.4 Comparison to Existing Methods

While opposition may sometimes employ methods from antithetic variates and low-discrepancy sequences, in general that is not the case. To elucidate the uniqueness of opposition in the following we distinguish it from these two methods.

3.2.4.1 Antithetic Variates

Suppose we desire to estimate $\xi = \mathbb{E}[f] = \mathbb{E}[Y_1, Y_2]$ with unbiased estimator

$$\hat{\xi} = \frac{Y_1 + Y_2}{2}. \tag{3.5}$$

If Y_1, Y_2 are i.i.d. then $var(\hat{\xi}) = var(Y_1 + Y_2)/2$. However, if $cov(Y_1, Y_2) < 0$ then the variance can be further reduced.

One method to accomplish this is through the use of a monotonic function h. Then, generate Y_1 as an i.i.d. value as before, but utilizing h our two variables are $h(Y_1)$ and $h(1 - Y_1)$, which are monotonic over interval [0,1]. And

$$\hat{\xi} = \frac{h(Y_1) + h(Y_2)}{2} \tag{3.6}$$

will be an unbiased estimator of $\mathbb{E}[f]$.

Opposition is similar in its selection of negatively correlated samples. However, in the antithetic approach there is no guideline to construct such a monotonic function, although such a function has been proven to exist [17]. Opposition provides various means to accomplish this, as well as to incorporate the idea into optimization while guaranteeing more desirable expected values and lower variance in the target function.

Further, opposition extends beyond the generation of solutions in random sampling-based algorithms. It can also be applied to algorithm behavior and can be used to relate concepts expressed linguistically, where we have not found evidence that antithetic variates have application.

3.2.4.2 Quasi-Randomness and Low-Discrepancy Sequences

These methods aim to combine the randomness from pseudorandom generators which select values i.i.d. with the advantages of generating points distributed in a grid-like fashion. The goal is to uniformly distribute values over the search space by achieving a low-discrepancy. However, this is achieved at the cost of statistical independence.

The discrepancy of a sequence is a measure of its uniformity and can be calculated via [6]:

$$D_N(X) = \sup_{I \in J} \left| \frac{|B \cap X|}{N} - \lambda_d(B) \right| \tag{3.7}$$

where λ_d is the d-dimensional Lebesque measure, $|B \cap X|$ is the number of points of $X = (x_1, ..., x_N)$ that fall into interval I, and J is the set of d-dimensional intervals defined as:

$$\prod_{i=1}^{d} [a_i, b_i) = \{x \in \Re^d : a_i \leq x_i \leq b_i\} \tag{3.8}$$

for $0 \leq a_i < b_i \leq 1$.

That is, the actual number of points within each interval for a given sample is close to the expected number. Such sequences have been widely studied in quasi-Monte Carlo methods [10].

Opposition may utilize low-discrepancy sequences in some situations. Though in general, low-discrepancy sequences are simply a means for attaining uniform distribution without regard to the correlation between the evaluations at these points. Further, opposition-based techniques simultaneously consider two points in order to smooth the evaluation function and improve performance of the sampling algorithm whereas quasi-random sequences often are concerned with many more points.

These methods have been applied to evolutionary algorithms where it was found that by a performance study of the different sampling methods such as Uniform, Normal, Halton, Sobol, Faure, and Low-Discrepancy is valuable only for low-dimensional ($d < 10$ and so non-highly-sparse) populations [11].

3.3 Algorithms

In this section we describe Differential Evolution (DE) and Population-¿ It seems, performance study of the different sampling methods such as Uniform, Normal, Halton, Sobol, Faure, and Low-Discrepancy [11] is valuable only for low-dimensional ($D < 10$ and so non-highly-sparse) populations. Learning (PBIL), which are the

parent algorithms for this study. We also describe the oppositional variants of these methods.

3.3.1 Differential Evolution

Differential Evolution (DE) was proposed by Price and Storn in 1995 [21]. It is an effective, robust, and simple global optimization algorithm [8]. DE is a population-based directed search method. Like other evolutionary algorithms, it starts with an initial population vector. If no preliminary knowledge about the solution space is available the population is randomly generated. Each vector of the initial population can be generated as follows [8]:

$$X_{i,j} = a_j + rand_j(0,1) \times (a_j - b_j); j = 1,2,...,d, \tag{3.9}$$

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 $[0,1]$.

Assume $X_{i,t}(i = 1,2,...,N_p)$ are candidate solution vectors in generation t and N_p : is the 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 ($DE/rand/1/bin$), the mutation, crossover, and selection operators are straightforwardly defined as follows:

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

$$V_{i,t} = X_{a,t} + F(X_{c,t} - X_{b,t}), \tag{3.10}$$

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 unique such that it is necessary for $N_p \geq 4$. $F \in [0,2]$ is a real constant which determines the amplification of the added differential variation of $X_{c,t} - X_{b,t}$. Larger values for F result in higher diversity in the generated population and lower values lead to faster convergence.

Crossover: By shuffling competing solution vectors DE utilizes the crossover operation to generate new solutions and also to increase the population diversity. 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:

$$U_{i,t} = (U_{1i,t}, U_{2i,t}, ..., U_{di,t}), \tag{3.11}$$

where,

$$U_{ji,t} = \begin{cases} V_{ji,t} & \text{if } rand_j(0,1) \leq C_r \vee j = k, \\ X_{ji,t} & \text{otherwise,} \end{cases} . \tag{3.12}$$

for $C_r \in (0,1)$ 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 mutated vector, $V_{ji,t}$. Most popular values for C_r are in the range of $(0.4, 1)$ [14].

Selection: This decides which vector ($U_{i,t}$ or $X_{i,t}$) should be a member of next (new) generation, $t+1$. 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.

3.3.2 Opposition-Based Differential Evolution

By utilizing opposite points, we can obtain fitter starting candidate solutions even when there is no a priori knowledge about the solution(s) according to:

1. Random initialization of population $P(N_P)$,
2. Calculate opposite population by

$$OP_{i,j} = \mathrm{a}_j + \mathrm{b}_j - \mathrm{P}_{i,j}, \tag{3.13}$$

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

   where $\mathrm{P}_{i,j}$ and $OP_{i,j}$ denote j^{th} variable of the i^{th} vector of the population and the opposite-population, respectively.
3. Selecting the N_p fittest individuals from $\{P \cup OP\}$ as the initial population.

The general ODE scheme also employs *generation jumping*, but it has not be used in this work in lieu of only population-initialization and sample generation.

3.3.3 Population-Based Incremental Learning

PBIL is a stochastic search which abstracts the population of samples found in evolutionary computation with a probability distribution for each variable of the solution [2]. At each generation a new sample population is generated based on the current probability distribution. The best individual is retained and the probability model is updated accordingly to reflect the belief regarding the final solution. The update rule is similar to that found in reinforcement learning.

A population is represented by probability matrix $\mathbf{M} := (m_{i,j})_{d \times c}$ which stores the probability distribution over each possible element in the solution. If considering a binary problem then solution $\mathbf{S} := (s_{i,j})_{d \times c} \in \{0,1\}$ and $m_{i,j} \in [0,1]$ is the probability element $s_{i,j} = 1$. For continuous problems probability distributions are used instead of a threshold value as is the case for discrete problems [18].

Learning consists of utilizing $\mathbf{M}$ to generate population $\mathscr{P}$ of k samples. After evaluation of each sample according to function f the "best" ($\mathbf{B}^*$) solution is retained and $\mathbf{M}$ is updated according to

$$\mathbf{M}_t = (1-\alpha)\mathbf{M}_{t-1} + \alpha\mathbf{B}^* \tag{3.14}$$

where $0 < \alpha < 1$ is the learning rate and $t \geq 1$ is the iteration. Initially, $m_{i,j} = 0.5$ to reflect that lack of prior information.

To abstract the crossover and mutation operators of evolutionary computation, PBIL employs a randomization of $\mathbf{M}$. Let $0 < \beta, \gamma < 1$ be the probability of mutation and degree of mutation, respectively. Then with probability β

$$m_{i,j} = (1-\gamma)m_{i,j} + \gamma \cdot random(0 \; or \; 1). \tag{3.15}$$

Algorithm 1 provides a summary of this approach.

Algorithm 1. Population-Based Incremental Learning [2]

```
{Initialize probabilities}
M_0 := (m_{i,j}) = 0.5
for t = 1 to ω do
    {Generate samples}
    G_1 = generate_samples(k,M_{t-1})

    {Find best sample}
    B* = select_best({B*} ∪ G_1)

    {Update M}
    M_t = (1 − α)M_{t-1} + αB*

    {Mutate probability vector}
    for i = 0...d and j = 0...c do
        if random(0,1) < β then
            m_{i,j} = (1 − γ)m_{i,j} + γ · random(0 or 1)
        end if
    end for
end for
```

3.3.4 Oppositional Population-Based Incremental Learning

The opposition-based version of PBIL (OPBIL) shown in Algorithm 2 employs the opposite concept to improve diversity within the sample generation phase. A direct effect on convergence rate is observed as a consequence of this mechanism. Further, OPBIL has an ability to escape local optima which estimation of distribution algorithms such as PBIL are prone to becoming trapped on [19]. The description provided here is brief and the interested reader is invited to read [29] for a more detailed description.

The general structure of the PBIL algorithm remains, however aside from the sampling procedure the update and mutation rules are altered to reflect a degrading

degree of opposition with respect to the number of iterations. That is, as the number of iterations $t \rightarrow \infty$ the amount two opposite solutions differ approaches 1 bit (w.r.t. Hamming distance).

Sampling is accompished using an opposite guessing strategy whereby half of the population R_1 is generated using probability matrix $\mathbf{M}$ and the other half is generated via a change in Hamming distance to a given element of R_1. The distance is calculated using an exponentially decaying function in the flavor of

$$\xi(t) = le^{(ct)}, \tag{3.16}$$

where l is the maximum number of bits in a guess and $c < 0$ is a user defined constant.

Updating of $\mathbf{M}$ is performed in lines 14-28. If a new global best solution has been discovered (i.e. $\eta = \mathbf{B}^*$), or with probability p_{amp} the sample best solution is used to focus the search, respectively. When no new optima have been discovered this strategy tends away from $\mathbf{B}^*$. The actual update is performed in line 16 and is based on a reinforcement learning update using the sample best solution. The degree to which $\mathbf{M}$ is updated is controlled by the user defined parameter $0 < \rho < 1$.

Should the above criteria for update fail, a decay of $\mathbf{M}$ with probability p_{decay} is attempted in line 17. The decay, performed in lines 21-27 slowly tends $\mathbf{M}$ away from $\mathbf{B}^*$. This portion of the algorithm has the intention to prevent convergence an aide in the exploration ability through small, smooth updates. Parameter $0 < \tau < 1$ is user defined where often $\tau \ll \rho$.

Equations in lines 11 and 12 were determined experimentally and no argument regarding their optimality is provided. Indeed, there likely exists many functions which will yield more desirable results. These have been decided because they tend to lead to a good behavior and outcome of PBIL.

3.4 Experimental Setup

In this section we provide a discussion of the image thresholding problem, and the application of evolutionary algorithms to solving it. Additionally, the evaluation measure we employ to grade the quality of a segmentation is presented. Parameter settings and problem representation are also given.

3.4.1 Evolutionary Image Thresholding

Image segmentation involves partitioning an image I into a set of segments with the goal of locating objects in the image which are sufficiently similar. Thresholding is a subset problem of image segmentation, with only 2 classes defined by whether a given pixel is above or below a specific threshold value ω. This task has numerous applications and several general segmentation algorithms have been proposed [33]. Due to the variety of image types there does not exist a single algorithm for segmenting all images optimally.

Algorithm 2. Pseudocode for the OPBIL algorithm

Require: Maximum iterations, ω
Require: Number of samples per iteration, k
1: {Initialize probabilities}
2: $\mathbf{M}_0 = m_{i..l} = 0.5$

3: **for** $t = 1$ to ω **do**
4: {Generate samples}
5: R_1 = generate_samples($k/2$,$\mathbf{M}$)
6: $\breve{R}_1$ = generate_opposites(R_1)

7: {Find best sample}
8: η = select_best($\{R_1 \cup \breve{R}_1\}$)
9: $\mathbf{B}^*$ = select_best($\mathbf{B}^*, \eta$)

10: {Compute probabilities}
11: $p_{amp}(\Delta) = 1 - e^{-b\Delta}$
12: $p_{decay}(\Delta; f(\mathbf{B}^*), f(\eta)) = \dfrac{1 - \frac{f(\mathbf{B}^*) - f(\eta)}{f(\mathbf{B}^*)}}{\sqrt{\Delta + 1}}$
13: {Update $\mathbf{M}$}
14: **if** $\eta = \mathbf{B}^*$ OR $random(0,1) < p_{amp}$ **then**
15: $\Delta = 0$
16: $\mathbf{M}_t = (1-\rho)\mathbf{M}_{t-1} + \rho\eta$
17: **else if** $random(0,1) < p_{decay}$ **then**
18: **if** $random(0,1) < p_{decay}$ **then**
19: use $\mathbf{B}^*$ in line 23 instead of η
20: **end if**
21: **for all** i, j each with probability $< p_{decay}$ **do**
22: **if** $\eta_{i,j} = \mathbf{B}^*_{i,j}$ **then**
23: $m_{i,j} = m_{i,j} \cdot \begin{cases} 1 - \tau \cdot random(0,1), & \text{if } \eta_{i,j} = 1, \\ 1 + \tau \cdot random(0,1), & \text{otherwise} \end{cases}$
24: **else**
25: $m_{i,j} = m_{i,j} \cdot \begin{cases} 1 + \tau \cdot random(0,1), & \text{if } \eta_{i,j} = 1, \\ 1 - \tau \cdot random(0,1), & \text{otherwise} \end{cases}$
26: **end if**
27: **end for**
28: **end if**
29: $\Delta = \Delta + 1$
30: **end for**

Many general purpose segmentation algorithms are histogram based and aim to discover a deep valley between two peaks, and setting ω equal to that value. However, many real world problems will have multimodal histograms and deciding which value (i.e. valley) will correspond to the best thresholing is not obvious. The difficulty is compounded by the fact that the relative size of peaks may be large (and the valley becomes hard to distinguish) or valleys could be very broad. Several algorithms have been proposed to overcome this [33]. Other methods based on information theory and other statistical methods have been proposed as well [13].

Typically, the problem of segmentation involves a high degree of uncertainty which makes solving the problem difficult. Stochastic searches such as evolutionary algorithms and population-based incremental learning often cope well with uncertainty in optimization, hence they provide an interesting alternative approach to traditional methods.

The main difficulty associated with the use of population-based methods is that they are computationally expensive due to the large amount of function calls required during the optimization process. One approach to minimizing uncertainty is by splitting the image into subimages which (hopefully) have characteristics allowing for an easy segmentation. Combining the subimages together then forms the entire segmented image, although this requires extra function calls to analyze each subimage. An important caveat is that the local image may represent a good segmentation, but may not be useful with respect to the image as a whole.

In this chaper we investigate thresholding with population-based techniques. Using ODE we do not perform any splitting into subimages and for OPBIL we split I into four equal-sized subregions, each having it's own threshold value. In both cases we require a single evaluation to perform the segmentation and we show that the opposition-based techniques reduce the required number of function calls.

As stated above, there exist many different segmentation algorithms. Further, numerous methods for evaluating the quality of a seqmentation have also been put forth [34]. In this paper we use a simple method which aims to minimize the discrepancy between the original $M \times N$ gray-level image I and its thresholded image T [31]:

$$\sum_{i=1}^{M} \sum_{j=1}^{N} |I_{i,j} - T_{i,j}| \tag{3.17}$$

where $|\cdot|$ represents the absolute value operation. Using different evaluations will change the outcome of the algorithm, however, the problem of segmentation in this manner nonetheless remains computationally expensive.

We use the images shown in Figure 3.3 to evaluate the algorithms. The first column represents the original image, then the gold and the third column corresponds to the approximate target image for ODE and OPBIL (i.e. the value-to-reach targets, discussed below). We show the gold image for completeness, it is not required in the experiments.

Both experiments employ a value-to-reach (VRT) stopping criteria which measures the time or function calls required to reach a specific value. The VTR values have been experimentally determined and are given in the following table. Due to

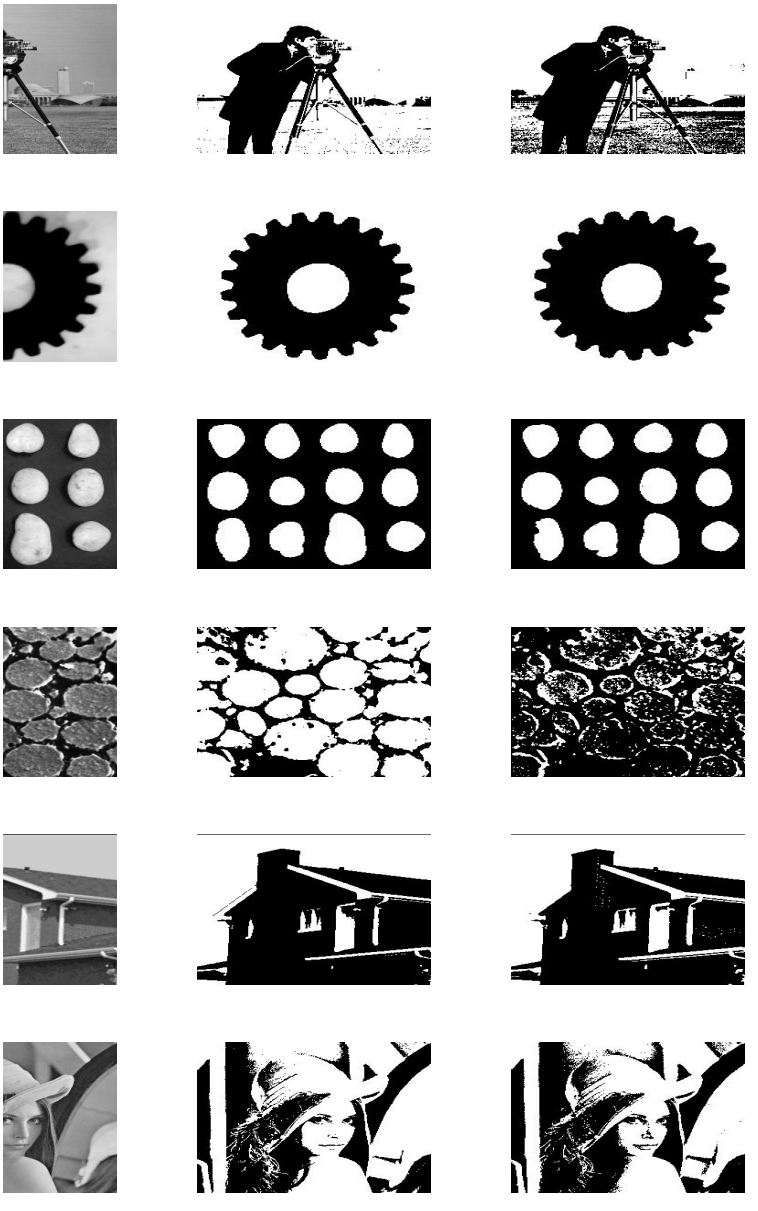

Fig. 3.3 The images used to benchmark the algorithms. The first column is the original gray-level image, the second is the gold and the third colum is the target image of the optimization within the required function calls

the respective algorithm ability of solving this problem, given the representation and bahavior of convergence these values differ for ODE and OPBIL.

Table 3.1 Value-to-reach (VRT) for O/DE and O/PBIL experiments

Image	O/DE	O/PBIL
1	19579	19850
2	3391	4925
3	7139	7175
4	19449	19850
5	19650	19700
6	22081	22700

3.4.2 Parameter Settings and Solution Representation

The ODE and OPBIL algorithms differ in that the former is a real optimization algorithm and the latter operates in the binary domain. Therefore, the solution representations will also differ and consequently directly affect the quality of results. However, the goal of this chapter is to show the ability of opposition to decrease the required number of function evaluations, and so fine-tuning aspects these algorithms is not the focus of this investigation.

ODE Settings

The differential evolution experiments follow standard enconding guidelines. The user-defined parameter settings were determined empirically as shown in Table 3.2.

Table 3.2 Parameter settings for differential evolution-based experiments

Parameter	Value
Population size	$N_p = 5$
Amplification factor	$F = 0.9$
Crossover probability	$Cr = 0.9$
Mutation strategy	DE/rand/1/bin
Maximum function calls	$\text{MAX}_{\text{NFC}} = 200$
Jumpring rate (no jumping)	$Jr = -1$

In order to maintain a reliable and fair comparison, these settings are kept unchanged for all conducted experiments for both DE and ODE algorithms.

OPBIL Settings

As stated above, OPBIL requires a binary solution representation. However, thresholding aims to discover an integer value $0 \leq T \leq 255$, to perform the segmentation operation of $I > T$. Additionally, we use an approach of splitting I into subimages $I_{1,\ldots,16}$ where each I_i is an equal sized square region of the original image.

Encoding was determined to be a matrix $\mathbf{R} := (r_{i,j})_{16\times 8}$ which corresponds to 16 subimages having a gray-level value $< 2^8 = 256$. Each row of $\mathbf{R}$ is converted to an integer which is used to segment the respective region of I. The extra regions increase problem difficulty as they result in more deceptive and multimodal problems.

Parameter settings for PBIL and OPBIL are as follows:

Table 3.3 Parameter settings for population-based incremental learning experiments

Parameter	Value
Maximum iterations	$t = 150$
Sample size	$k = 24$
PBIL Only	
Learning rate	$\alpha = 0.35$
Mutation probability	$\beta = 0.15$
Mutation degree	$\gamma = 0.25$
OPBIL Only	
Update frequency control	$b = 0.1$
Learning rate	$\rho = 0.25$
Probability decay	$\tau = 0.0005$

3.5 Experimental Results

Using the test images and parameter settings stated above we show the ability of oppositional concepts to decrease the required function calls. Only the results for OPBIL are more detailed due to space limitations, but similar behavior should be observed in ODE. All results presented correspond to the average of 30 runs, unless otherwise noted.

3.5.1 ODE

Table 3.4 presents a summary of the results obtained regarding function calls for ODE versus DE. Except for image 2, we show an decrease in function calls for all images. Images 4 and 5 have statistically significant improvements with respect to the decreased number of function calls, using a t-test at 0.9 confidence level. Further except for image 6, we show a lower standard deviation which indicates higher reliability in the results.

Computing the overall mean function calls we show an improvement of 322-277=45 function calls. This equates to an average of 45/6 = 7.5 saved function calls per image. This implies a savings of $322/277 \approx 1.16$, indicating approximately 16% less function calls. For expensive optimization problems this can correspond to a great amount of savings with respect to algorithm run-time.

Table 3.4 Summary results for DE vs. ODE with respect to required function calls. μ and σ correspond to the mean and standard deviation of the subscripted algorithm, respectively

Image	μ_{DE}	σ_{DE}	μ_{ODE}	σ_{ODE}
1	74	41	60	34
2	32	20	35	16
3	42	23	37	19
4	74	36	**54**	30
5	47	37	45	22
6	63	26	**46**	31
Total	322		277	

3.5.2 OPBIL

Table 3.5 shows the expected number of iterations (each iteration has 24 function calls) to attain the value-to-reach given in Table 3.1. In all cases OPBIL reaches its goal in fewer iterations that PBIL, where results for images 2,5,6 are found to be statistically significant using a t-test at a 0.9 confidence interval. Additionally, in all cases we find a lower standard deviation indicating a more reliable behavior for OPBIL.

Overall we have 444-347=97 saved iterations using OPBIL, which is an average of 16*24=384 function calls per image. The approximate savings is $444/347 \approx 1.28$ which is about a 28% improvement in required iterations.

Table 3.5 Summary results for PBIL vs. OPBIL with respect to required iterations calls. μ and σ correspond to the mean and standard deviation of the subscripted algorithm, respectively

Image	μ_{PBIL}	σ_{PBIL}	μ_{OPBIL}	σ_{OPBIL}
1	62	19	53	12
2	80	25	**65**	9
3	61	12	60	5
4	47	14	40	10
5	68	13	**53**	9
6	128	21	**76**	14
Total	444		347	

In the following we analyze the correlation and distance for each sample per iteration. This is to examine whether the negative correlation and larger distance properties between a guess and its opposite are found in the sample. If true, we have supported (although not confirmed) the hypothesis that the observed improvements can be due to these characteristics.

Figure 3.4 shows the averaged correlation $cor(R_1^t, \breve{R}_1)$, for randomly generated R_1^t and respective opposites $\breve{R}_1^t$ at iteration t. The solid line corresponds to OPBIL and the dotted line is PBIL, respectively. These plots show that OPBIL indeed has a lower correlation (with respect to evaluation function) than PBIL (where we generate R_1 as above, and let $\breve{R}_1$ also be randomly generated). In all cases the correlation is much stronger for PBIL (noting that if the algorithm reaches the VTR then we set the correlation to 0).

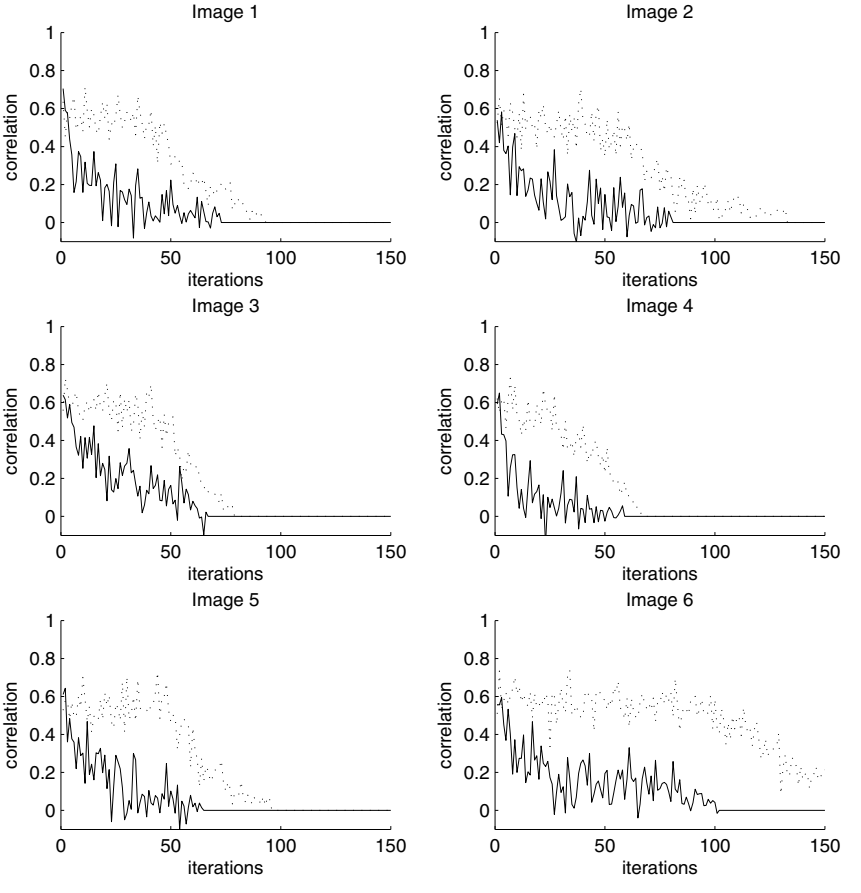

Fig. 3.4 Sample mean correlation over 30 trials for PBIL (dotted) versus OPBIL (solid). We find OPBIL indeed yields a lower correlation than PBIL

We also examine the mean distance,

$$\bar{g} = \frac{2}{k} \sum_{i=1}^{k/2} g(R^t_{1,i}, \breve{R}^t_{1,i}) \tag{3.18}$$

which computes the fitness-distance between the i^{th} guess $R_{1,i}$ and its opposite $\breve{R}_{1,i}$ at iteration t, which is shown in Figure 3.5. The distance for PBIL is relatively low throughout the 150 iterations, gently decreasing as the algorithm converges. However, as a consequence of OPBIL's ability to mainain diversity the distances between samples increases during the early stanges of the search and similarily rapidly decreases. Indeed, this implies the lower correlation shown above.

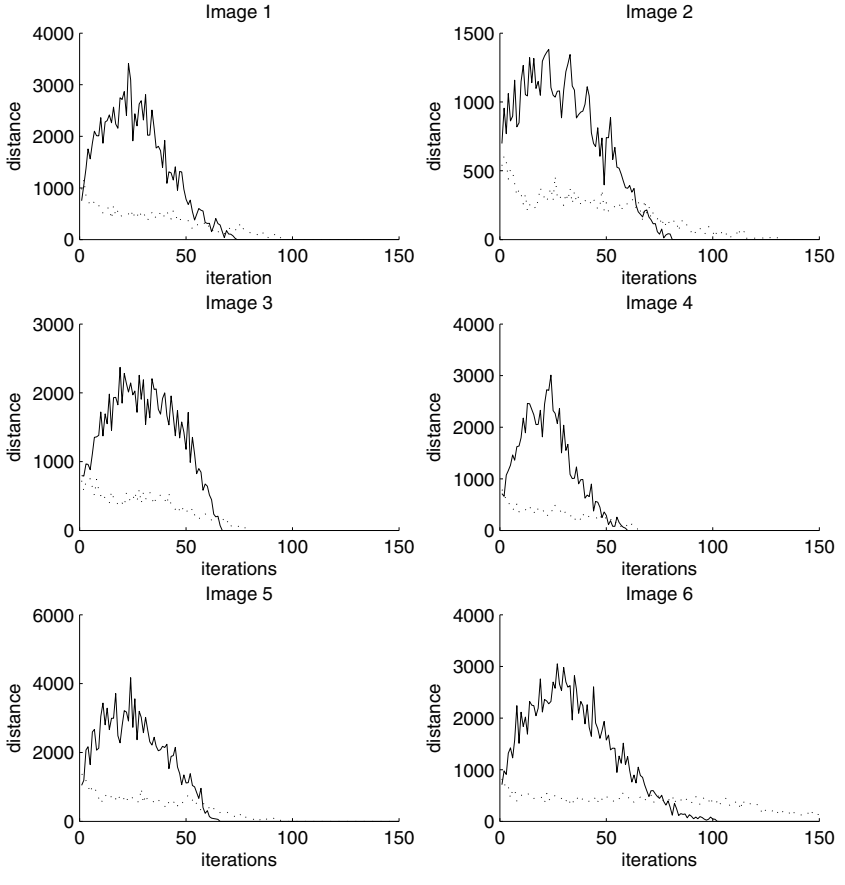

Fig. 3.5 Sample mean distance over 30 trials for samples of PBIL (dotted) versus OPBIL (solid). We find OPBIL indeed yields a higher distance between paired samples than PBIL

The final test is to examine the standard deviation (w.r.t. evaluation function) of the distance between samples, given in Figure 3.6. Both algorithms have similarly formed plots with respect to this measure, reflecting the convergence rate of the respective algorithms. It seems as though the use of opposition aides by infusing diversity early during the search and quickly focuses once a high quality optima is found. Conversely, the basic PBIL does not include this bias, therefore convergence is less rapid.

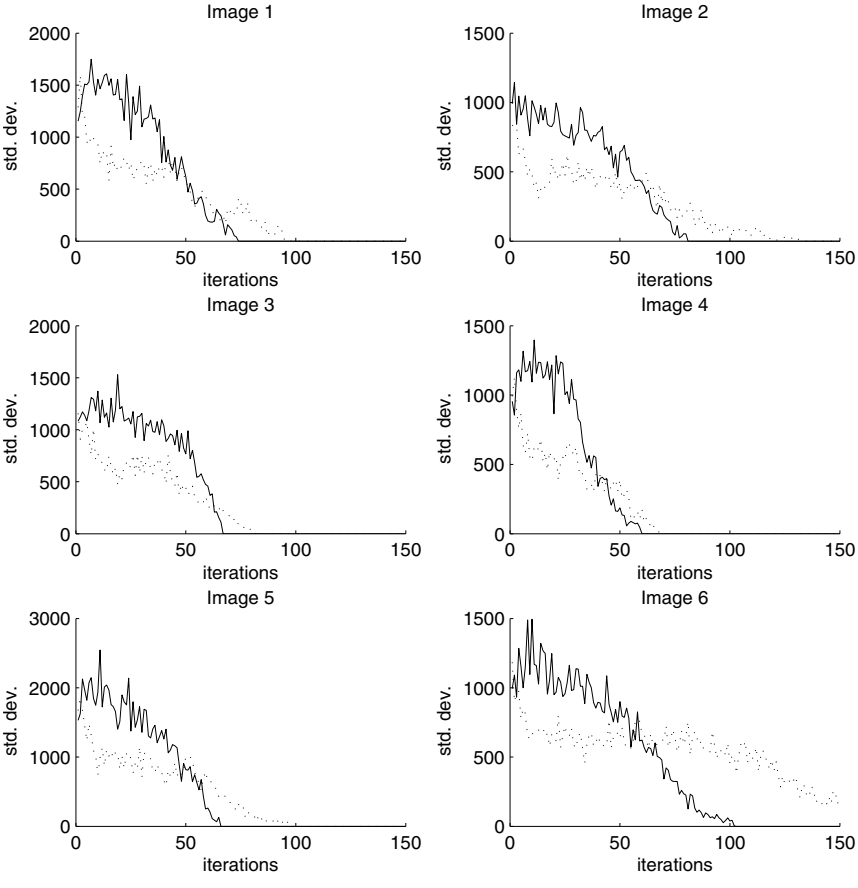

Fig. 3.6 Sample mean standard deviations over 30 trials for samples of PBIL (dotted) versus OPBIL (solid)

3.6 Conclusion

In this chapter we have discussed the application of opposition-based computing techniques to reducing the required number of function calls. Firstly, a brief

introduction to the underlying concepts of opposition were given, along with conditions under which opposition-based methods should be successful. A comparison to similar concepts of antithetic variates and quasi-random/low-discrepancy sequences made obvious the uniqueness of our method.

Two recently proposed algorithms, ODE and OPBIL were briefly introduced and the manner in which opposition is used to improve their parent algorithms, DE and PBIL was given, respectively. The manner in which opposites are used in both cases differed, but the underlying concepts are the same.

Using the expensive optimization problem of image thresholding as a benchmark, we examine the ability of ODE and PBIL to lower the required function calls to reach the pre-specified target value. It was found that both algorithms reduce the expected number of function calls, ODE by approximately 16% (function calls) and OPBIL by 28% (iterations), respectively. Further, concentrating on OPBIL, we show the hypothesized lower correlation and higher fitness-distance measures for a quality opposite mapping.

Our results are very promising, however their requires future work in various regards. Firstly, a further theoretical basis for opposition and choosing opposite mappings is needed. This could possibly lead to general strategies of implementation when no prior knowledge is available. Further application to different real-world problems is also desired.

Acknowledgements

This work has been partially supported by Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Bai, F., Wu, Z.: A novel monotonization transformation for some classes of global optimization problems. Asia-Pacific Journal of Operational Research 23(3), 371–392 (2006)
2. Baluja, S.: Population Based Incremental Learning - A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Tech. rep., Carnegie Mellon University, CMU-CS-94-163 (1994)
3. Engelbrecht, A.: Fundamentals of Computational Swarm Intelligence. Wiley, Chichester (2005)
4. Glover, F., Laguna, M.: Tabu Search. Kluwer, Dordrecht (1997)
5. Goldberg, D.E., Horn, J., Deb, K.: What makes a problem hard for a classifier system? Tech. rep. In: Collected Abstracts for the First International Workshop on Learning Classifier Systems (IWLCS 1992), NASA Johnson Space (1992)
6. Niederreiter, H.: Random Number Generation and Quasi-Monte Carlo Methods. Society for Industrial and Applied Mathematics (1992)
7. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press (1975)
8. Price, K., Storn, R., Lampinen, J.A.: Differential Evolution: A Practical Approach to Global Optimization. Springer, Heidelberg (2005)

9. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science 220(4598), 671–680 (1983)
10. Lemieux, C.: Monte Carlo and Quasi-Monte Carlo Sampling. Springer, Heidelberg (2009)
11. Maaranen, H., Miettinen, K., Penttinen, A.: On intitial populations of genetic algorithms for continuous optimization problems. Journal of Global Optimization 37(3), 405–436 (2007)
12. Montgomery, J., Randall, M.: Anti-pheromone as a tool for better exploration of search space. In: Third International Workshop, ANTS, pp. 1–3 (2002)
13. O'Gormam, L., Sammon, M., Seul, M. (eds.): Practical Algorithms for Image Analysis. Cambridge University Press, Cambridge (2008)
14. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: Opposition-based differential evolution. IEEE Transactions on Evolutionary Computation 12(1), 64–79 (2008)
15. Rahnamayn, S., Tizhoosh, H.R., Salama, S.: Opposition-based Differential Evolution Algorithms. In: IEEE Congress on Evolutionary Computation, pp. 7363–7370 (2006)
16. Rahnamayn, S., Tizhoosh, H.R., Salama, S.: Opposition-based Differential Evolution Algorithms for Optimization of Noisy Problems. In: IEEE Congress on Evolutionary Computation, pp. 6756–6763 (2006)
17. Rubinstein, R.: Monte Carlo Optimization, Simulation and Sensitivity of Queueing Networks. Wiley, Chichester (1986)
18. Sebag, M., Ducoulombier, A.: Extending Population-Based Incremental Learning to Continuous Search Spaces. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 418–427. Springer, Heidelberg (1998)
19. Shapiro, J.: Diversity loss in general estimation of distribution algorithms. In: Parallel Problem Solving in Nature IX, pp. 92–101 (2006)
20. Shokri, M., Tizhoosh, H.R., Kamel, M.: Opposition-based Q(lambda) Algorithm. In: IEEE International Joint Conference on Neural Networks, pp. 646–653 (2006)
21. Storn, R., Price, K.: Differential evolution- a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 341–359 (1997)
22. Tizhoosh, H.R.: Reinforcement Learning Based on Actions and Opposite Actions. In: International Conference on Artificial Intelligence and Machine Learning (2005)
23. Tizhoosh, H.R.: Opposition-based Reinforcement Learning. Journal of Advanced Computational Intelligence and Intelligent Informatics 10(4), 578–585 (2006)
24. Tizhoosh, H.R., Ventresca, M. (eds.): Oppositional Concepts in Computational Intelligence. Springer, Heidelberg (2008)
25. Toh, K.: Global optimization by monotonic transformation. Computational Optimization and Applications 23(1), 77–99 (2002)
26. Ventresca, M., Tizhoosh, H.R.: Improving the Convergence of Backpropagation by Opposite Transfer Functions. In: IEEE International Joint Conference on Neural Networks, pp. 9527–9534 (2006)
27. Ventresca, M., Tizhoosh, H.R.: Opposite Transfer Functions and Backpropagation Through Time. In: IEEE Symposium on Foundations of Computational Intelligence, pp. 570–577 (2007)
28. Ventresca, M., Tizhoosh, H.R.: Simulated Annealing with Opposite Neighbors. In: IEEE Symposium on Foundations of Computational Intelligence, pp. 186–192 (2007)
29. Ventresca, M., Tizhoosh, H.R.: A diversity maintaining population-based incremental learning algorithm. Information Sciences 178(21), 4038–4056 (2008)

30. Ventresca, M., Tizhoosh, H.R.: Numerical condition of feedforward networks with opposite transfer functions. In: IEEE International Joint Conference on Neural Networks, pp. 3232–3239 (2008)
31. Weszka, J., Rosenfeld, A.: Threshold evaluation techniques. IEEE Transactions on Systems, Man and Cybernetics 8(8), 622–629 (1978)
32. Wu, Z., Bai, F., Zhang, L.: Convexification and concavification for a general class of global optimization problems. Journal of Global Optimization 31(1), 45–60 (2005)
33. Yoo, T. (ed.): Insight into Images: Principles and Practice for Segmentation, Registration, and Image Analysis. AK Peters (2004)
34. Zhang, H., Fritts, J., Goldman, S.: Image segmentation evaluation: A survey of unsupervised methods. Computer Vision and Image Understanding 110, 260–280 (2008)