Improve Self-Adaptive Control Parameters in Differential Evolution Algorithm for Complex Numerical Optimization Problems

A DISSERTATION SUBMITTED TO
GRADUATE SCHOOL OF ENGINEERING AND SCIENCE OF
SHIBAURA INSTITUTE OF TECHNOLOGY

by

BUI NGOC TAM

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF ENGINEERING

SEPTEMBER 2015

Acknowledgments

This dissertation is a result of research that has been performed at the Hasegawa laboratory, College of Systems Engineering and Science, Shibaura Institute of Technology, Japan, under the supervision of Prof. Hiroshi Hasegawa. Completion of this doctoral dissertation was possible with the support of several people. I would like to express my sincere gratitude to all of them.

First of all, I am heartily thankful to my supervisor, Prof.Hiroshi Hasegawa, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. I am sure it would have not been possible without his help.

I would like to acknowledge the financial, academic and technical support, gradate school section and Student Affairs Section especially Ms.Yabe in Omiya campus of the Shibaura Institute of Technology.

I would like to thank all other members of Hasegawa laboratory for their contributions to all kinds of discussions on various topics, and their support with respect. The group has been a source of friendships as well as good advice and collaboration

I would like to thank to my wife Nguyen Thi Hien for her personal support and great patience and my family at all times. My parents, brother and sister have given me their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice.

Japan, September 2015

BUI NGOC TAM

Abstract

Memetic Algorithms (MA) is effective algorithms to obtain reliable and accurate solutions for complex continuous optimization problems. Nowadays, high dimensional optimization problems are an interesting field of research. To solve complex numerical optimization problems, researchers have been looking into nature both as model and as metaphor for inspiration. A keen observation of the underlying relation between optimization and biological evolution led to the development of an important paradigm of computational intelligence for performing very complex search and optimization.

Evolutionary Computation uses iterative process, such as growth or development in a population that is then selected in a guided random search using parallel processing to achieve the desired end. Nowadays, the field of nature-inspired metaheuristics is mostly continued by the Evolution Algorithms (EAs) (e.g., Genetic Algorithms (GAs), Evolution Strategies (ESs), and Differential Evolution (DE) etc.) as well as the Swarm Intelligence algorithms (e.g., Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC), etc.). Also the field extends in a broader sense to include self-organizing systems, artificial life, memetic and cultural algorithms, harmony search, artificial immune systems, and learnable evolution model.

In this thesis, we propose the improvement self-adaptive for controlling parameters in differential evolution (ISADE) and investigate the hybridization of a local search algorithm with an evolution algorithm (H-MNS_ISADE), which are the Nelder-Mead simplex method (MNS) and differential evolution (DE), for Complex numerical optimization

problems. This approach hybrid integrate differential evolution with Nelder-Mead simplex method technique is a component based on where the DE algorithm is integrated with the principle of Nelder-Mead simplex method to improve the neighborhood search of the each particle in H-MNS_ISADE. By using local information of MNS and global information obtained from DE population, the exploration and exploitation abilities of H-MNS_ISADE algorithm are balanced. All the algorithms applied to the some benchmark functions and compared based on some different metrics.

This dissertation includes three main points - firstly, we propose the improvement self-adaptive for controlling parameters in differential evolution (ISADE) to solve large scale optimization problems, to reduce calculation cost, and to improve stability of convergence towards the optimal solution; secondly, new algorithms (ISADE) is applied to several numerical benchmark tests, constrained real parameter optimization and trained artificial neural network to evaluate its performancem, and finally, we introduce the hybridization of a local search algorithm with an evolution algorithm (H-MNS_ISADE), which are the Nelder-Mead simplex method (MNS) and differential evolution (DE);

Contents

List of Figures

LIST OF FIGURES

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Optimal Systems Design

It is no exaggeration to say that optimization is everywhere, from engineering design to business planning and from the routing of the Internet to holiday planning. we are trying to achieve certain objectives or to optimize something such as profit, quality and time. As resources, time and money are always limited in real-world applications, we have to find solutions to optimally use these valuable resources under various constraints. For several decades, global optimization has received a wide attraction from researchers, mathematicians as well as professionals in the field of Operations Research (OR) and Computer Science (CS). However, global optimization problems, in almost all fields of research and real-world applications, have many different challenging features such as high non-linearity, non-convexity, non-continuity, non-differentiability, and/or multimodality. Therefore, classical nonlinear optimization techniques have difficulties or have always failed in dealing with complex high dimensional global optimization problems. As a result, the challenges mentioned above have motivated researchers to design and improve many kinds of efficient, effective and robust algorithms that can reach a high quality solution with low computational cost and high convergence performance.

1.2 Optimal Design of Complex Mechanical Systems

As follow [51]. The concept of design was born the first time an individual created an object to serve human needs. Today design is still the ultimate expression of the art and science of engineering. From the early days of engineering, the goal has been to improve the design so as to achieve the best way of satisfying the original need, within the available means.

The design process can be described in many ways, but we can see immediately that there are certain elements in the process that any description must contain: a recognition of need, an act of creation, and a selection of alternatives. Traditionally, the selection of the "best" alternative is the phase of design optimization. In a traditional description of the design phases, recognition of the original need is followed by a technical statement of the problem (problem definition), the creation of one or more physical configurations (synthesis), the study of the configuration's performance using engineering science (analysis), and the selection of "best" alternative (optimization). The process concludes with testing of the prototype against the original need.

Such sequential description, though perhaps useful for educational purposes, cannot describe reality adequately since the question of how a "best" design is selected within the available means is pervasive, influencing all phases where decisions are made.

So what is design optimization?
We defined it loosely as the selection of the "best" design within the available means. This may be intuitively satisfying; however, both to avoid ambiguity and to have an operationally useful definition we ought to make our understanding rigorous and, ideally, quantifiable. We may recognize that a rigorous definition of "design optimization" can be reached if we answer the questions:

1. How do we describe different designs?
2. What is our criterion for "best" design?
3. What are the "available means"?

The first question was addressed in the previous discussion on design models, where a design was described as a system defined by design variables, parameters, and constants. The second question was also addressed in the previous section in the discussion on decision-making models where the idea of "best" design was introduced and the criterion for an optimal design was called an objective. The objective function is sometimes called a "cost" function since minimum cost often is taken to characterize the "best" design. In general, the criterion for selection of the optimal design is a function of the design variables in the model.

We are left with the last question on the "available means." Living, working, and designing in a finite world obviously imposes limitations on what we may achieve. Brushing aside philosophical arguments, we recognize that any design decision will be subjected to limitations imposed by the natural laws, availability of material properties, and geometric compatibility. On a more practical level, the usual engineering specifications imposed by the clients or the codes must be observed. Thus, by "available means" we signify a set of requirements that must be satisfied by any acceptable design. Once again we may observe that these design requirements may not be uniquely defined but are under the same limitations as the choice of problem objective and variables. In addition, the choices of design requirements that must be satisfied are very intimately related to the choice of objective function and design variables.

As an example, consider again the shaft design (shown in Figure. 1.1) . If we choose minimum weight as objective and diameter d as the design variable, then possible specifications are the use of a particular material, the fixed length, and the transmitted loads and revolutions. The design requirements we may impose are that the maximum stress should not exceed the material strength and perhaps that the maximum deflection should not surpass a limit imposed by the need for proper meshing of mounted gears. Depending on the kind of bearings used, a design requirement for the slope of the shaft deflection curve at the supporting ends may be necessary. Alternatively, we might choose to maximize rigidity, seeking to minimize the maximum deflection as an objective. Now the design requirements might change to include a limitation in the space D available for mounting, or even the maximum weight that we can tolerate in a "lightweight"

construction. We resolve this issue by agreeing that the design requirements to be used are relative to the overall problem definition and might be changed with the problem formulation. The design requirements pertaining to the current problem definition we will call design constraints. We should note that design constraints include all relations among the design variables that must be satisfied for proper functioning of the design.

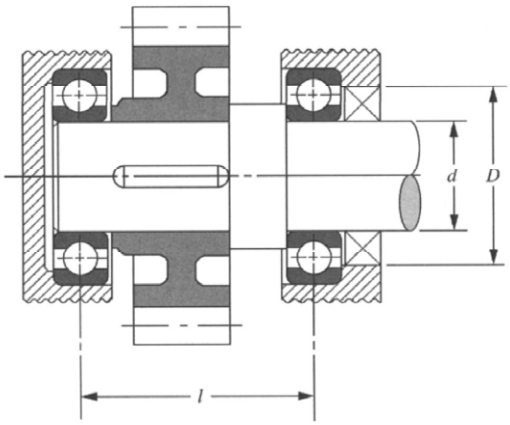

Figure 1.1: Sketch of a shaft design.[51]

So what is design optimization?
Informally, but rigorously, we can say that design optimization involves:

1. The selection of a set of variables to describe the design alternatives.

2. The selection of an objective (criterion), expressed in terms of the design variables, which we seek to minimize or maximize.

3. The determination of a set of constraints, expressed in terms of the design variables, which must be satisfied by any acceptable design.

4. The determination of a set of values for the design variables, which minimize (or maximize) the objective, while satisfying all the constraints.

Formulation of the optimization problem

Mathematically speaking, it is possible to write most optimization problems in the generic form The optimization problem is formulated in this section. The design

variable, objective function and constraint condition are defined as follows:

$$\textit{Objective function: } f_l(x) \rightarrow Minimize, \quad (l = 1, \ldots, L) \tag{1.1}$$

$$\textit{Equality constraint functions: } h_j(x) = 0, \quad (j = 1, \ldots, J) \tag{1.2}$$

$$\textit{Inequality constraint functions: } g_k(x) \leq 0, \quad (k = 1, \ldots, K) \tag{1.3}$$

$$\textit{Range of design variables: } x_i^{lb} \leq x_i \leq x_i^{ub} \tag{1.4}$$

Here the components x_i of x are called design or decision variables, and they can be real continuous, discrete or the mixed of these two.

The functions $f_l(x)$ where $(l = 1, \ldots, L)$ are called the objective function or simply cost functions, and in the case of L = 1, there is only a single objective. $x^{lb} = [x_1^{lb}, \ldots, x_D^{lb}]$, $x^{ub} = [x_1^{ub}, \ldots, x_D^{ub}]$, and D denote the lower boundary condition vectors, upper boundary condition vectors, and number of design variable vectors, respectively. J and K are the number of equality and inequality constraints respectively. h_j and g_k are linear or nonlinear real-value functions respectively.

In a rare but extreme case where there is no objective at all, there are only constraints. Such a problem is called a feasibility problem because any feasible solution is an optimal solution.

If we try to classify optimization problems according to the number of objectives, then there are two categories: single objective $L = 1$ and multiobjective $L > 1$. Multiobjective optimization is also referred to as multicriteria or even multi-attributes optimization in the literature. In real-world problems, most optimization tasks are multiobjective. Though the algorithms we will discuss in this book are equally applicable to multiobjective optimization with some modications, we will mainly place the emphasis on single objective optimization problems.

Similarly, we can also classify optimization in terms of number of constraints $J+K$. If there is no constraint at all $J = K = 0$, then it is called an unconstrained optimization problem. If $K = 0$ and $J \geq 1$, it is called an equality-constrained problem, while $J = 0$ and $K \geq 1$ becomes an inequality-constrained problem. It is worth pointing out that in some formulations in the optimization literature, equalities are not explicitly included, and only inequalities are included. This is because an equality can be written as two inequalities. For example $h(x) = 0$ is equivalent to $h(x) \leq 0$ and $h(x) \geq 0$.

We can also use the actual function forms for classication. The objective functions can be either linear or nonlinear. If the constraints h_j and g_k are all linear, then it becomes a linearly constrained problem. If both the constraints and the objective functions are all linear, it becomes a linear programming problem. Here programming has nothing to do with computing programming, it means planning and/or optimization. However, generally speaking, all f_l , h_j and g_k are nonlinear, we have to deal with a nonlinear optimization problem.

Thus we talk about equality and inequality constraints given in the form of equal to zero and less than or equal to zero. For example, in our previous shaft design, suppose we used a hollow shaft with outer diameter d_o , inner diameter d_i, and thickness t. These quantities could be viewed as design variables satisfying the equality constraint

$$d_o = d_i + 2t \ , \tag{1.5}$$

which can be rewritten as

$$d_o - d_i - 2t = 0 \ , \tag{1.6}$$

so that the constraint function is

$$h(d_o, d_i, t) = d_o - d_i - 2t \ , \tag{1.7}$$

We could also have an inequality constraint specifying that the maximum stress does not exceed the strength of the material, for example,

$$\sigma_{max} \leq S \ , \tag{1.8}$$

where S is some properly defined strength (i.e., maximum allowable stress). However, σ_{max} should be expressed in terms of for simplicity, we can write

$$\sigma_{max} = \tau_{max} = M_t(d_o/2)/J \ , \tag{1.9}$$

where M_t is the torsional moment and J is the polar moment of inertia,

$$J = (\pi/32)\left(d_o^4 - d_i^4\right) \ , \tag{1.10}$$

At this point we may view eq. (1.9) and eq. (1.10) as additional equality constraints with σ_{max} and J being additional design variables. Note that M_t would be a design parameter. Thus, we can rewrite them as follows:

$$\sigma_{max} - S \leq 0 \ , \tag{1.11}$$

$$\sigma_{max} - M_t(d_o/2J) = 0 \ , \tag{1.12}$$

$$J - (\pi/32)\left(d_o^4 - d_i^4\right) = 0 \ , \tag{1.13}$$

so that we have one inequality and two equality constraints corresponding to eq. (1.8). We could also eliminate σ_{max} and J and get

$$16M_t d_o/\pi\left(d_o^4 - d_i^4\right) - S \leq 0 \ , \tag{1.14}$$

that is, just one inequality constraint. This implies that σ_{max} and J were considered *intermediate variables* that with the formulation eq. (1.14) will disappear from the model statement. The above operation from eq. (1.11) to eq. (1.14) is a *model transformation* and it must be always performed judiciously so that the problem resulting from the transformation is *equivalent* to the original one and usually easier to solve. A strict definition of equivalence is difficult. Normally, we simply mean that the solution set of the transformed model is the same as that of the original model.

1.3 Constraints and Challenges

As mentioned in section.1.2. A natural and important question is how to incorporate the constraints (both inequality and equality constraints). There are mainly three ways to deal with constraints: direct approach, Lagrange multipliers, and penalty method.

Direct approach intends to find the feasible regions enclosed by the constraints. This is often difficult, except for a few special cases. Numerically, we can generate a potential solution, and check if all the constraints are satisfied. If all the constraints are met, then it is a feasible solution, and the evaluation of the objective function can be carried out. If one or more constraints are not satisfied, this potential solution is discarded, and a new solution should be generated. We then proceed in a similar manner. As we can expect, this process is slow and inefficient. A better approach is to incorporate the constraints so as to formulate the problem as an unconstrained one. The method of Lagrange multiplier has rigorous mathematical basis, while the penalty method is simple to implement in practice.

1.3.1 Method of Lagrange Multipliers

The method of Lagrange multipliers converts a constrained problem to an unconstrained one [23, 52]. For example, if we want to minimize a function:

$$\textit{minimize: } f(x), \quad x = (x_1, \ldots, x_D)^T \subset \Re^D \tag{1.15}$$

subject to multiple nonlinear equality constraints

$$g_j(x) = 0, \quad (j = 1, \ldots, M) \tag{1.16}$$

We can use M Lagrange multipliers $\lambda_j (j = 1, \ldots, M)$ to reformulate the above problem as the minimization of the following function

$$L(x, \lambda_j) = f(x) + \sum_{j=1}^{M} \lambda_j g_j(x) \tag{1.17}$$

The optimality requires that the following stationary conditions hold

$$\frac{\partial L}{\partial x_i} = \frac{\partial f}{\partial x_i} + \sum_{j=1}^{M} \lambda_j \frac{\partial g_j}{\partial x_i}, \quad (i = 1, \ldots, D) \tag{1.18}$$

and

$$\frac{\partial L}{\partial \lambda_i} = g_j = 0, \quad (j = 1, \ldots, M) \tag{1.19}$$

These $M+D$ equations will determine the D components of x and M Lagrange multipliers. As $\dfrac{\partial L}{\partial g_i} = \lambda_j$, we can consider λ_j as the rate of the change of the quantity $L(x, \lambda_j)$ as a functional of g_j .

Now let us look at a simple example

$$maximize : f = u^{2/3} v^{1/3} \tag{1.20}$$

subject to

$$3u + v = 9 \tag{1.21}$$

First, we write it as an unconstrained problem using a Lagrange multiplier λ, and we have

$$L = u^{2/3} v^{1/3} + \lambda(3u + v - 9) \tag{1.22}$$

The conditions to reach optimality are

$$\frac{\partial L}{\partial u} = \frac{2}{3} u^{-1/3} v^{1/3} + 3\lambda = 0, \qquad \frac{\partial L}{\partial v} = \frac{1}{3} u^{2/3} v^{-2/3} + \lambda = 0 \tag{1.23}$$

and

$$\frac{\partial L}{\partial \lambda} = 3u + v - 9 = 0 \tag{1.24}$$

The first two conditions give $2v = 3u$, whose combination with the third condition leads to

$$u = 2, \qquad v = 3.$$

Thus, the maximum of f_* is $12^{1/3}$

Here we only discussed the equality constraints. For inequality con- straints, things become more complicated. We need the so-called Karush- Kuhn-Tucker conditions.

Let us consider the following, generic, nonlinear optimization problem

$$\textit{minimize: } f(x), \quad x = (x_1, \ldots, x_D)^T \subset \Re^D \tag{1.25}$$

subject to multiple nonlinear constraints

$$\phi_i(x) = 0, \quad (i = 1, \ldots, M) \tag{1.26}$$

$$\psi_j(x) \leqslant 0, \quad (j = 1, \ldots, N) \tag{1.27}$$

If all the functions are continuously differentiable, at a local minimum x^* , there exist constants $\lambda_1, \ldots, \lambda_M$ and $\mu_0, \mu_1, ..., \mu_N$ such that the following KKT optimality conditions hold

$$\mu_0 \nabla f(x_*) + \sum_{i=1}^{M} \lambda_i \nabla \phi_i(x_*) + \sum_{j=1}^{N} \mu_j \nabla \psi_j(x_*) = 0 \tag{1.28}$$

and

$$\psi_j(x_*) \leqslant 0, \qquad \mu_j \psi_j(x_*) = 0, \qquad (j = 1, \ldots, N) \tag{1.29}$$

where

$$\mu_j \geq 0, \qquad (j = 1, \ldots, N) \tag{1.30}$$

The last non-negativity conditions hold for all μ_j, though there is no constraint on the sign of λ_i.

The constants satisfy the following condition

$$\sum_{j=1}^{N} \mu_j + \sum_{i=1}^{M} |\lambda_i| \geq 0 \tag{1.31}$$

This is essentially a generalized method of Lagrange multipliers. However, there is a possibility of degeneracy when $\mu_0 = 0$ under certain conditions. There are two possibilities:
1) There exist vectors $\lambda = (\lambda_1^*, \ldots, \lambda_M^*)^T$ such that above equations hold, or
2) All the vectors $\nabla\phi_1(x_*), \nabla\phi_1(x_*), \ldots, \nabla\phi_N(x_*)$ are linearly independent, and in this case, the stationary conditions $\dfrac{\partial L}{\partial x_*}$ do not necessarily hold. As the second case is a special case, we will not discuss this further.

The condition $\mu_j\psi_i(x_*) = 0$ in eq.(1.29) is often called the complementarity condition or complementary slackness condition. It either means $\mu_j = 0$ or $\psi_j(x_*) = 0$. The later case $\psi_j(x_*) = 0$ for any particular j means the inequality becomes tight, and thus becoming an equality. For the former case $\mu_j = 0$, the inequality for a particular j holds and is not tight; however, $\mu_j = 0$ means that this corresponding inequality can be ignored. There- fore, those inequalities that are not tight are ignored, while inequalities which are tight become equalities; consequently, the constrained problem with equality and inequality constraints now essentially becomes a modified constrained problem with selected equality constraints. This is the beauty of the KKT conditions. The main issue remains to identify which inequality becomes tight, and this depends on the individual optimization problem.

The KKT conditions form the basis for mathematical analysis of non- linear optimization problems, but the numerical implementation of these conditions is not easy, and often inefficient. From the numerical point of view, the penalty method is more straightforward to implement.

1.3.2 Penalty Method

The first is the now classical penalty approach developed by Fiacco and McCormick [17]. For a nonlinear optimization problem with equality and inequality constraints, a common method of incorporating constraints is the penalty method. For the optimization problem

$$\textit{minimize: } f(x), \quad x = (x_1, \ldots, x_D)^T \subset \Re^D \tag{1.32}$$

subject to multiple nonlinear constraints

$$\phi_i(x) = 0, \quad (i = 1, \ldots, M) \tag{1.33}$$

$$\psi_j(x) \leqslant 0, \quad (j = 1, \ldots, N) \tag{1.34}$$

the idea is to dene a penalty function so that the constrained problem is transformed into an unconstrained problem. Now we define

$$\Pi(x, \mu, \nu) = f(x) + \sum_{i=1}^{M} \mu_i \phi_i^2(x) + \sum_{j=1}^{N} \nu_j \psi_j^2(x) \tag{1.35}$$

where $\mu_i \gg 1$ and $\nu_j \geqslant 0$ which should be large enough, depending on the solution quality needed.

As we can see, when an equality constraint it met, its effect or contribution to is zero. However, when it is violated, it is penalized heavily as it increases significantly. Similarly, it is true when inequality constraints become tight or exact. For the ease of numerical implementation, we should use index functions H to rewrite above penalty function as

$$\Pi(x, \mu, \nu) = f(x) + \sum_{i=1}^{M} \mu_i H_i[\phi_i(x)] \phi_i^2(x) + \sum_{j=1}^{N} \nu_j H_j[\psi_j(x)] \psi_j^2(x) \tag{1.36}$$

Here $H_i[\phi_i(x)]$ and $H_j[\psi_j(x)]$ are index functions.

More specically, $H_i[\phi_i(x)] = 1$ if $\phi_i(x) \neq 0$, and $H_i = 0$ if $\phi_i(x) = 0$. Similarly, $H_j[\psi_j(x)] = 0$ if $\psi_j(x) \leq 0$ is true, while $H_j = 1$ if $\psi_j(x) > 0$. In principle, the numerical accuracy depends on the values of μ_i and ν_j which should be reasonably large. But how large is large enough? As most computers have a machine precision of $\varepsilon = 2^{52}2.2 \times 10^{16}$, μ_i and ν_j should be close to the order of 10^15 . Obviously, it could cause numerical problems if they are too large.

In addition, for simplicity of implementation, we can use $\mu = \mu_i$ for all i and $\nu = \nu_j$ for all j. That is, we can use a simplified

$$\Pi(x, \mu, \nu) = f(x) + \mu \sum_{i=1}^{M} H_i[\phi_i(x)]\phi_i^2(x) + \nu \sum_{j=1}^{N} H_j[\psi_j(x)]\psi_j^2(x) \tag{1.37}$$

In general, for most applications, μ and ν can be taken as 10^{10} to 10^{15} . We will use these values in our implementation.

Sometimes, it might be easier to change an equality constraint to two inequality constraints, so that we only have to deal with inequalities in the implementation. This is because $g(x) = 0$ is always equivalent to $g(x) \leq 0$ and $g(x) \geq 0$ (or $g(x) \leq 0$).

1.3.3 Step Size in Random Walks

As random walks are widely used for randomization and local search, a proper step size is very important [70]. In the generic equation:

$$x^{t+1} = x^t + s\epsilon_t \tag{1.38}$$

ϵ_t is drawn from a standard normal distribution with zero mean and unity standard deviation. Here the step size s determines how far a random walker (e.g., an agent or particle in metaheuristics) can go for a fixed number of iterations.

If s is too large, then the new solution x^{t+1} generated will be too far away from the old solution (or more often the current best). Then, such a move is

unlikely to be accepted. If s is too small, the change is too small to be significant, and consequently such search is not efficient. So a proper step size is important to maintain the search as efficient as possible.

From the theory of simple isotropic random walks, we know that the average distance r traveled in the d-dimension space is

$$r^2 = 2dDt \tag{1.39}$$

where $D = s^2/2\tau$ is the effective diffusion coefficient. Here s is the step size or distance traveled at each jump, and τ is the time taken for each jump. The above equation implies that

$$s^2 = \frac{\tau}{r}^2 td \tag{1.40}$$

For a typical length scale L of a dimension of interest, the local search is typically limited in a region of $L/10$. That is, $r = L/10$. As the iterations are discrete, we can take $\tau = 1$. Typically in metaheuristics, we can expect that the number of generations is usually $t = 100$ to 1000, which means that

$$s \approx \frac{r}{\sqrt{td}} = \frac{L/10}{td} \tag{1.41}$$

For $d = 1$ and $t = 100$, we have $s = 0.01L$, while $s = 0.001L$ for $d = 10$ and $t = 1000$. As step sizes could differ from variable to variable, a step size ratio s/L is more generic. Therefore, we can use $s/L = 0.001$ to 0.01 for most problems.

1.4 Motivation and Objects

Evolutionary Algorithms (EAs) have been widely applied to solve complex numerical optimization problems, especially the multi-peak problems with multi-dimensions. The most popular EA, Genetic Algorithm (GA) [18, 26, 27, 28] has been applied to various multi-peak optimization problems, and its validity has

been reported by many researchers. Digalakis and Margaritis presented a review and experimental results on major benchmark functions which are used for performance and control of GA [9].

In 1992, Marco Dorigo finished his PhD thesis on optimization and natural algorithms, in which he described his innovative work on ant colony optimization (ACO). This search technique was inspired by the swarm intelligence of social ants using pheromone as a chemical messenger. Then, in 1992, John R. Koza of Stanford University published a treatise on genetic programming which laid the foundation of a whole new area of machine learning, revolutionizing computer programming. As early as in 1988, Koza applied his first patent on genetic programming. The basic idea is to use the genetic principle to breed computer programs so as to gradually produce the best programs for a given type of problem.

Slightly later in 1995, another significant progress is the development of the particle swarm optimization (PSO) by American social psychologist James Kennedy, and engineer Russell C. Eberhart. Loosely speaking, PSO is an optimization algorithm inspired by swarm intelligence of fish and birds and by even human behavior. The multiple agents, called particles, swarm around the search space starting from some initial random guess. The swarm communicates the current best and shares the global best so as to focus on the quality solutions. Since its development, there have been about 20 different variants of particle swarm optimization techniques, and have been applied to almost all areas of tough optimization problems. There is some strong evidence that PSO is better than traditional search algorithms and even better than genetic algorithms for many types of problems, though this is far from conclusive.

In around 1995 and later in 1997, R. Storn and K. Price developed their vector-based evolutionary algorithm, called differential evolution (DE) [61, 62], and this algorithm proves more efficient than genetic algorithms in many applications.

At the turn of the 21st century, things became even more exciting. First, Zong Woo Geem et al in 2001 developed the harmony search (HS) algorithm, which has been widely applied in solving various optimization problems such as

water distribution, transport modeling and scheduling. In 2004, S. Nakrani and C. Tovey proposed the honey bee algorithm and its application for optimizing Internet hosting centers, which followed by the development of a novel bee algorithm by D. T. Pham et al in 2005 and the artificial bee colony (ABC) by D.Karaboga in 2005 [38]. In 2008, Xin-She Yang developed the firey algorithm (FA) [68] . Quite a few research articles on the firey algorithm then followed, and this algorithm has attracted a wide range of interests. In 2009, Xin-She Yang at Cambridge University, UK, and Suash Deb at Raman College of Engineering, India, introduced an efficient cuckoo search (CS) algorithm [72], and it has been demonstrated that CS is far more effective than most existing metaheuristic algorithms including particle swarm optimization. In 2010, the author Xin-She Yang developed a bat-inspired algorithm [71] for continuous optimization, and its efficiency is quite promising.

To reduce the cost, to improve stability and get more accurate, a strategy that combines global and local search methods becomes necessary. As for this strategy, current researchers have proposed various methods. One of the popular approach is a combination of global search ability of GAs with local search ability of Simulated Annealing (SA) [54]. As a pioneering research, Mahfoud and Goldberg have proposed Parallel Recombinative Simulated Annealing (PRSA) that applied SA to a selection of GA [45]. Later, Uehara et al. have introduced metropolis loop process of SA to an elite strategy in GA process [66, 67]. Hiroyasu et al. have proposed Parallel SA using Genetic crossover (PSA/ANGA) [24, 46]. These hybrid methods have been applied to major benchmark functions and have been reported to be valid. They are believed to be both locally and globally efficient. However, the major multi-peak benchmark functions for multi-dimensions, i.e., 20 dimensional or more Rastrigin (RA) and Griewank (GR) functions, require about 10^6 function calls for arriving at an optimal solution. Moreover, when the optimal problem exhibits a dependence on design variable vectors (DVs) and the steepness of the objective function is small in the feasible space of DVs, it is difficult to obtain an optimal solution [22].

Various optimization methodologies are proposed to overcome these difficulties [4, 19, 20, 21, 22, 48, 49, 50, 64]. In Memetic Algorithms (MAs) [4, 19, 48, 49,

50, 64], for instance, Ong and Keane has proposed meta-Lamarckian learning [50] that improves the search ability for multi-peak functions with multi-dimensions by introducing a human expert judgment, where local search methods are used. Additionally, fast Adaptive Memetic Algorithm (FAMA) has been proposed in [4]. In the FAMA, coordination and choosing of local search method are dynamically controlled by means of a measurement of fitness diversity over the individuals of the population. On the other hand, Hasegawa et al. have proposed a hybrid meta-heuristic method (HMH) by reflecting recognition of dependence relations among design variables automatically, and have reported the effectiveness of this method [21, 22]. The HMH needs to switch from the SA to the intuitive method, direct search using the learning result of the dependency of a DV, just before convergence to improve the local search ability of the optimal solution environs. These methodologies need to choose suitably a best local search method from various local search methods for combining with a global search method within the optimization process. Furthermore, since genetic operators are employed for a global search method within these algorithms, DVs which are renewed via a local search are encoded into its genes many times at its GA process. These certainly have the potential to break its improved chromosomes via gene manipulation by GA operators, even if these approaches choose a proper survival strategy.

To solve these problems and maintain the stability of the convergence towards an optimal solution for multi-modal optimization problems with multiple dimensions. In this dissertation, we focus to some motivation as below:

Firstly, automatic control parameter in differential evolution algorithm by proposed a new improvement of self-adaptive strategy for controlling parameters in differential evolution algorithm (ISADE). The differential evolution (DE) algorithm has been used in many practical cases and has demonstrated good convergence properties. It has only a few control parameters as number of particles (NP), scaling factor (F) and crossover control (CR), which are kept fixed throughout the entire evolutionary process. However, these control parameters are very sensitive to the setting of the control parameters based on their experiments. The value of control parameters depend on the characteristics of each

objective function, so we have to tune their value in each problem that mean it will take too long time to perform.

Secondly, new algorithms (ISADE) is applied to several numerical benchmark problems, constrained real parameter optimization and Training Artificial Feed-forward Neural Network to evaluate its performance.

Finally, improve local search ability of differential evolution algorithm by proposed propose Hybrid Improved Self-adaptive Differential Evolution and Nelder-Mead Simplex Method for Solving Constrained Real-Parameters.

1.5 Contributions

The overall objectives of these methodologies proposed in this dissertation are to solve large scale optimization problems, to reduce calculation cost, and to improve stability of convergence towards the optimal solution. Therefore, the approach that can lead to statistically significantly superior to other techniques is especially considered in this dissertation. The contributions of this dissertation are as follows

Firstly, we present a new version of the DE algorithm for obtaining self-adaptive control parameter settings that show good performance on numerical benchmark problems

Secondly, we proposed a new method of Training Artificial Feed-forward Neural Network.

Finally, integrated local search ability to DE algorithm.

1.6 Outline

The dissertation begins with the introduction the optimal systems design for complex numerical optimization problems. Then, the specific challenges and constraints for optimization techniques are discussed.

Chapter 2 describes a brief introduction to a metaheuristic algorithm for global optimization, Evolutionary Computing, such as GAs, DE, ABC , PSO, ACO, etc.

Chapter 3 proposes Improve Self-Adaptive Control Parameters in Differential Evolution to solve large scale optimization problems.

Chapter 4 proposes Hybrid Improved Self-adaptive Differential Evolution and Nelder-Mead Simplex Method for Solving Constrained Real-Parameters.

In Chapter 5, we introduce the method of Training Artificial Feed-forward Neural Network using Modification of Differential Evolution Algorithm.

Finally, the dissertation ends with conclusion, discussion, and future work in Chapter 6.

Chapter 2

Metaheuristic Algorithms for Global Optimization

Computational Intelligence (CI) is a set of nature-inspired computational methodologies and approaches to address complex real-world problems to which traditional approaches, i.e., first principles modeling or explicit statistical modeling, are ineffective or infeasible. Evolutionary computation (EC) is a subfield of artificial intelligence (AI) (more particularly CI) that involves continuous optimization and combinatorial optimization problems. Its algorithms can be considered global optimization methods with a metaheuristic or stochastic optimization character and are mostly applied for black box problems, often in the context of expensive optimization.

2.1 Introduction bimimetic

Bio-mimetic is the science of studying functional systems in nature and implementing or borrowing these features for human technology. Bio-mimetic can aid in the solving of new design problems or in the optimization of current technologies. Since natural systems are highly optimized for their purposes/functionality due to the constraint of survivability, it makes sense for human engineers to seek design hints from preexisting natural solutions.

Bio-mimetic is the intentional imitation of natural design. In some cases, human engineers have made inventions independently of nature, and only in retrospect we realized the similarities in design solutions. A well-cited example of this phenomenon is the similarity between certain bacterial flagellum and the outboard rotary motor. Both systems use very similar techniques for achieving the same functional effect, but this is coincidental and not an example of Bio-mimetic. Designing a molecular motor to deal with molecular dynamics by copying the bacterial flagellum, however, would be an example of Bio-mimetic.

Examples of Bio-mimetic include:

1. Identifying and implementing the technology that a leaf uses to harness energy
2. Making stronger, more elastic materials like the web of a spider
3. Designing miniaturized flying devices as found in millions of insects
4. Barbs on weed seeds as the inspiration for Velcro
5. Looking to the Rhinoceros horn to develop self-healing material that is both compressively and laterally strong
6. Implementing computer systems after the neural networks in our brains

In the optimization field, there are many applications of bio-mimetic for solving optimal problems. We can list some typically examples of bio-mimetic as: Genetic algorithms (GAs) [28], proposed John Holland in the early 1975s, are search algorithms based on the mechanics of selection and nature genetics. Differential evolution (DE) was proposed by Storn and Price [61]. Artificial Bee Colony (ABC), first introduced by Karaboga in 2005 [38], is a novel swarm intelligence (SI) algorithm that was inspired by the foraging behavior of honeybees. Particle Swarm Optimization (PSO) [39] was proposed by Kennedy and Eberhart in 1995 and so on

2.2 A brief introduction of Evolutionary Algorithm

2.2.1 What is an Evolutionary Algorithm (EA)

Evolutionary Algorithms (EAs) are stochastic optimization techniques based on the principles of natural evolution. The standpoint of EAs is essentially practical: using ideas from natural evolution in order to solve a certain problem. Let us focus on optimization and see how this goal can be achieved. Evolutionary algorithms operate on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. At each generation, the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics creates a new set of approximations. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation. Evolutionary algorithms model natural processes, such as selection, recombination, mutation, migration, locality and neighborhood. Evolutionary algorithms work on populations of individuals instead of single solutions. In this way, the search is performed in a parallel manner.

2.2.2 Components of Evolutionary Algorithms

In this section Evolutionary Algorithms are showed in detail. EAs have a number of components, procedures or operators that must be specified in order to define a particular EA. The most important components are:

- Representation (definition of individuals)
- Evaluation function (or fitness function)
- Population
- Parent selection mechanism

- Variation operators, recombination (crossover) and mutation
- Survival selection mechanism (replacement)

Furthermore, to obtain a running algorithm the initialization procedure and a termination condition must be defined.

The combined application of variation and selection generally leads to improving fitness values in consecutive populations. It is easy to view such an evolutionary process as optimization by iteratively generating solutions with increasingly better values. Alternatively, evolution it is often seen as a process of adaptation. From this perspective, the fitness is not seen as an objective function to be optimized, but as an expression of environmental requirements. Matching these requirements more closely implies an increased viability, reflected in a higher number of offspring. The evolutionary process makes the population increasingly better at being adapted to the environment.

The general scheme of an evolutionary algorithm is shown in Fig. 2.1 in a pseudocode fashion. It is important to note that many components of evolutionary algorithms are stochastic. During selection, fitter individuals have a higher chance to be selected than less fit ones, but typically even the weak individuals have a chance to become a parent or to survive. For recombination of individuals the choice of which pieces will be recombined is random. Similarly for mutation, the pieces that will be mutated within a candidate solution, and the new pieces replacing them, are chosen randomly. Fig. 2.2 shows a diagram.

It is easy to see that this scheme falls in the category of generate and test algorithms. The evaluation (fitness) function represents a heuristic estimation of solution quality and the search process is driven by the variation and the selection operators. Evolutionary Algorithms (EAs) posses a number of features that can help to position them within in the family of generate and test methods:

- EAs are population based, i.e., they process a whole collection of candidate solutions simultaneously,

- EAs mostly use recombination to mix information of more candidate solutions into a new one,
- EAs are stochastic.

```
BEGIN
    INITIALISE population with random candidate solutions;
    EVALUATE each candidate
    REPEAT UNTIL ( TERMINATION CONDITION is satisfied) DO
        SELECT parents;
        RECOMBINE pairs of parents;
        MUTATE the resulting offspring;
        EVALUATE new candidates;
        SELECT individuals for the next generation;
    END DO
END
```

Figure 2.1: The general scheme of Evolutionary Algorithm.

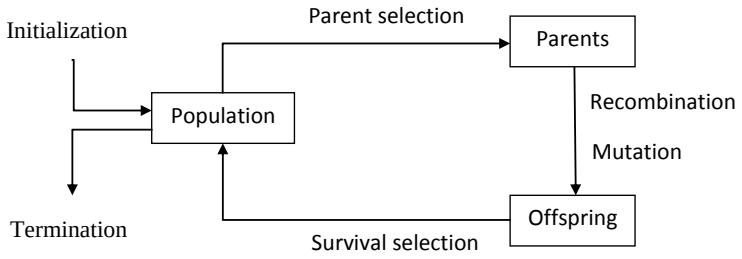

Figure 2.2: Flow-chart of Evolutionary Algorithm.

2.3 Simulated Annealing (SA)

One of the earliest and yet most popular metaheuristic algorithms is simulated annealing (SA) [41], which is a trajectory-based, random search technique for global optimization. It mimics the annealing process in material processing when a metal cools and freezes into a crystalline state with the minimum energy and larger crystal size so as to reduce the defects in metallic structures. The annealing process involves the careful control of temperature and its cooling rate, often called annealing schedule.

2.3.1 Annealing and Boltzmann Distribution

Since the first development of simulated annealing by Kirkpatrick, Gelatt and Vecchi in 1983 [41], SA has been applied in almost every area of optimization. Unlike the gradient-based methods and other deterministic search methods which have the disadvantage of being trapped into local minima, the main advantage of simulated annealing is its ability to avoid being trapped in local minima. In fact, it has been proved that simulated annealing will converge to its global optimality if enough randomness is used in combination with very slow cooling. Essentially, simulated annealing is a search algorithm via a Markov chain, which converges under appropriate conditions.

Metaphorically speaking, this is equivalent to dropping some bouncing balls over a landscape, and as the balls bounce and lose energy, they settle down to some local minima. If the balls are allowed to bounce enough times and lose energy slowly enough, some of the balls will eventually fall into the globally lowest locations, hence the global minimum will be reached.

The basic idea of the simulated annealing algorithm is to use random search in terms of a Markov chain, which not only accepts changes that improve the objective function, but also keeps some changes that are not ideal. In a minimization problem, for example, any better moves or changes that decrease the value of the objective function f will be accepted; however, some changes that increase f will also be accepted with a probability p. This probability p, also called the transition probability, is determined by

$$p = e^{-\frac{\Delta E}{k_B T}}, \tag{2.1}$$

where k_B is the Boltzmanns constant, and for simplicity, we can use k to denote k_B because $k = 1$ is often used. T is the temperature for controlling the annealing process. ΔE is the change of the energy level. This transition probability is based on the Boltzmann distribution in statistical mechanics.

The simplest way to link ΔE with the change of the objective function Δf is to use

$$\Delta E = \gamma \Delta f \ , \tag{2.2}$$

where γ is a real constant. For simplicity without losing generality, we can use $k_B = 1$ and $\gamma = 1$. Thus, the probability p simply becomes

$$p(\Delta f, T) = e^{-\Delta f/T} \ , \tag{2.3}$$

Whether or not we accept a change, we usually use a random number r as a threshold. Thus, if $p > r$, or

$$p = e^{-\Delta f/T} > r \ , \tag{2.4}$$

the move is accepted.

2.3.2 SA Algorithm

The simulated annealing algorithm can be summarized as the pseudo code shown in Fig. 2.3

In order to find a suitable starting temperature T_0 , we can use any information about the objective function. If we know the maximum change $\max(\triangle f)$ of the objective function, we can use this to estimate an initial temperature T_0 for a given probability p_0 . That is

$$T_0 \approx -\frac{max(\bigtriangledown f)}{lnp_0} \ , \tag{2.5}$$

if we do not know the possible maximum change of the objective function, we can use a heuristic approach. We can start evaluations from a very high temperature (so that almost all changes are accepted) and reduce the temperature quickly until about 50% or 60% of the worse moves are accepted, and then use this temperature as the new initial temperature T_0 for proper and relatively slow cooling.

Simulated Annealing Algorithm

Objective function $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, ..., x_p)^T$
Initialize initial temperature T_0 *and initial guess* $\boldsymbol{x}^{(0)}$
Set final temperature T_f *and max number of iterations* N
Define cooling schedule $T \mapsto \alpha T$, $(0 < \alpha < 1)$
while ($T > T_f$ *and* $n < N$)
    Move randomly to new locations: $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{\epsilon}$ *(random walk)*
    Calculate $\Delta f = f_{n+1}(\boldsymbol{x}_{n+1}) - f_n(\boldsymbol{x}_n)$
    Accept the new solution if better
    if *not improved*
        Generate a random number r
        Accept if $p = \exp[-\Delta f/T] > r$
    end if
    Update the best $\boldsymbol{x}_*$ *and* f_*
    $n = n + 1$
end while

Figure 2.3: Simulated annealing algorithm.

For the final temperature, it should be zero in theory so that no worse move can be accepted. However, if $T_f 0$, more unnecessary evaluations are needed. In practice, we simply choose a very small value, say, $T_f = 10^{-10} \sim 10^{-5}$, depending on the required quality of the solutions and time constraints.

2.4 Genetic Algorithms (GA)

Genetic Algorithm (GA) [18, 26, 27, 28] is one of the most popular evolutionary algorithms. The most common type of genetic algorithm works like this: a population is created with a group of individuals created randomly. The individuals in the population are then evaluated. The evaluation function is provided by the programmer and gives the individuals a score based on how well they perform at the given task. Two individuals are then selected based on their fitness, the higher the fitness, the higher the chance of being selected. These individuals then "reproduce" to create one or more offspring, after which the offspring are mutated

randomly. This continues until a suitable solution has been found or a certain number of generations have passed, depending on the needs of the programmer.

1. Initialization

   Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Traditionally, the population is generated randomly, covering the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

2. Selection

   Selection is the stage of a GA in which individual genomes are chosen from a population for later breeding (recombination or crossover).

   The most common type - fitness proportionate selection (also known as roulette-wheel selection), individuals are given a probability of being selected that is directly proportionate to their fitness. Two individuals are then chosen randomly based on these probabilities and produce offspring.

3. Crossover

   Crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next.

4. Mutation

   Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next.

   The purpose of mutation in GAs is to allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution.

5. Termination

   This generational process is repeated until a termination condition has been reached.

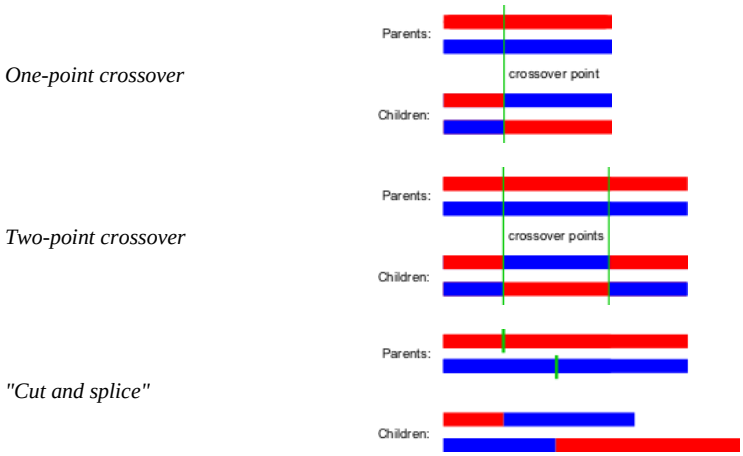

Figure 2.4: GA crossover operation.

Simple generational GA pseudocode

- Choose the initial population of individuals
- Evaluate the fitness of each individual in that population
- Repeat on this generation until termination:

  - Select the best-fit individuals for reproduction
  - Breed new individuals through crossover and mutation operations to give birth to offspring
  - Evaluate the individual fitness of new individuals
  - Replace least-fit population with new individuals

2.5 Differential Evolution (DE) Algorithm

Differential evolution algorithm was first proposed by R.Storn and K.Price [61]. DE is similar to other EAs particularly GA in the sense that it uses the same evolutionary operators such as selection, recombination, and mutation (a simple cycle of stages presented in Fig. 2.5). However the significant difference is that DE uses distance and direction information from the current population to guide the

search process. The performance of DE depends on the manipulation of target vector and difference vector in order to obtain a trial vector.

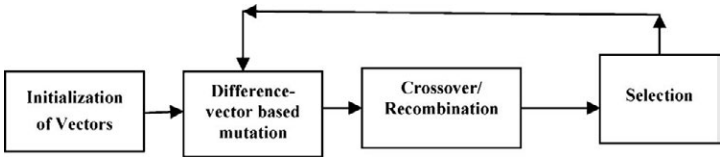

Figure 2.5: Main stages of DE algorithm.

1. Mutation

   The main scheme in DE becomes mutation operator. For each target vector $X_{i,G}$ in a D-dimensional search space, the process to obtain a mutant vector as follow:

   DE/rand/1

   $$V_{i,G} = X_{r_1,G} + F \cdot (X_{r_2,G} - X_{r_3,G}) \ , \tag{2.6}$$

   where $r_1,\ r_2,\ r_3 \in [1, 2, \ldots, NP]$ are mutually exclusive randomly chosen integers with a initiated population of NP, and all are different from the base index i. G denotes subsequent generations, and $F > 0$ is a scaling factor which controls the amplification of differential evolution.

   The process is illustrated on a 2-D parameter space (showing constant cost contours of an arbitrary objective function) in Fig. 2.6.

2. Crossover

   To enhance the potential diversity of the population, a crossover operation is introduced shown in Fig. 2.7. The donor vector exchanges its components with the target vector to form the trial vector:

   $$U_{ij,G+1} = \begin{cases} V_{ij,G+1}, (rand_j \leq CR) \ or \ (j = j_{rand}) \\ X_{ij,G+1}, (rand_j \geq CR) \ and \ (j \neq j_{rand}) \end{cases} \ , \tag{2.7}$$

   where $j = [1, 2, \ldots, D]$; $rand_j \in [0.0, 1.0]$; CR is the crossover probability takes value in the range [0.0,1.0], and $j_{rand} \in [1, 2, \ldots, D]$ is the randomly chosen index.

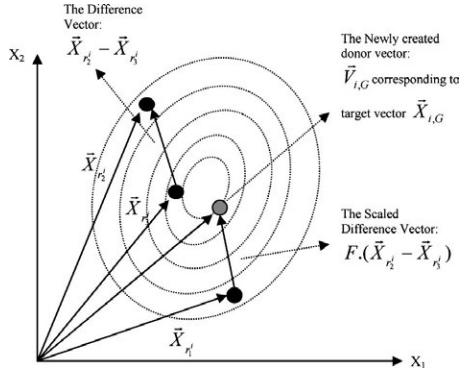

Figure 2.6: Illustrating a simple DE mutation scheme in 2-D parametric space.[61]

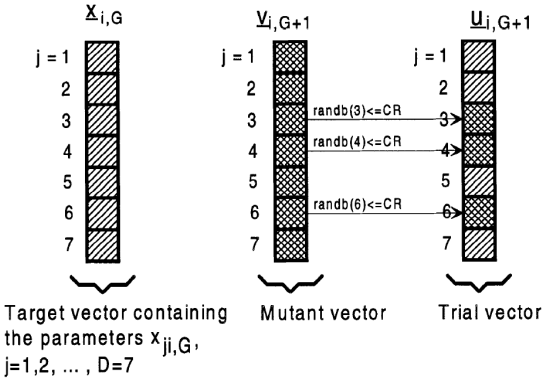

Figure 2.7: Illustration of the crossover process with D = 7.[61]

3. Selection

   To determine whether the target vector or the trial vector survives to the next generation, selection is performed. The selection operation is described as:

$$X_{i,G+1} = \begin{cases} U_{i,G}, & f(U_{i,G}) \leq f(X_{i,G}) \\ X_{i,G}, & f(U_{i,G}) > f(X_{i,G}) \end{cases} . \tag{2.8}$$

2.6 Artificial Bee Colony Algorithm (ABC)

ABC is a novel swarm intelligence (SI) algorithm, which was inspired by the foraging behavior of honeybees. ABC was first introduced by Karaboga in 2005 [38].

ABC is simple in concept, easy to implement, and it uses few control parameters, and hence, it has attracted the attention of researchers and has been used widely for solving many numerical [12], [11] and practical engineering optimization problems [10], [1].

There are two types of artificial bees:
- **First,** the employed bees that are currently exploiting a food source.
- **Second,** the unemployed bees that are continually looking for a food source.

Unemployed bees are divided into scout bees that search around the nest and onlooker bees that wait at the nest and establish communication with employee bees.

The tasks of each type of bee are as follows:
- **Employed Bee:** A bee that continues to forage a food source that it visited previously is known as an employed bee.
- **Onlooker Bee:** A bee that waits in the dance area to make a decision about a food source is known as an onlooker bee.
- **Scout Bee:** When a nectar food source is abandoned by bees, it is replaced with new a food source found by scout bees. If a position cannot be improved further after a predetermined number of cycles, the food source is assumed to be abandoned. The predetermined number of cycles is an important control parameter for ABC, which is known as the "*limit*" before abandonment.

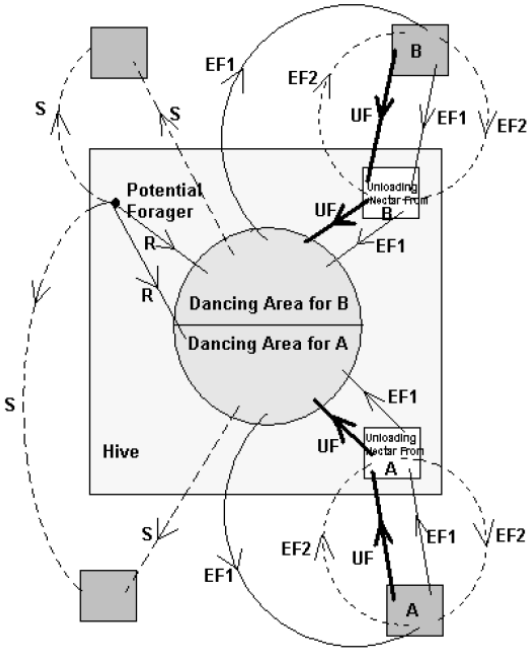

Figure 2.8: Behavior of honeybees foraging for nectar.[38]

To better understand the basic behavioral characteristics of foragers, Karaboga [38] used figure 2.8. In this example, we have two discovered food sources: A and B. At the start, a potential forager is an unemployed forager. This bee will have no knowledge of the food sources around the nest. There are the following two possible options for this bee.

- The bee can become a scout and start searching around the nest spontaneously for a food source owing to some internal motivation or possible external clues (S in Figure 2.8).
- It can become a recruit after observing waggle dances and start exploiting a food source (R in Fig. 2.8).

After locating the food source, the bee memorizes the location and immediately starts exploiting it. Thus, the bee will become an employed forager. The foraging bee collects a load of nectar from the source and returns to the hive, before unloading the nectar in a food store. After unloading the food, the bee has the following three options:

- It becomes a non-committed follower after abandoning the food source (UF).
- It dances and recruits nest mates before returning to the same food source (EF1).
- It continues to forage at the food source without recruiting other bees (EF2).

It is important to note that not all bees begin foraging simultaneously. Experiments have confirmed that new bees begin foraging at a rate proportional to the difference between the eventual total number of bees and the number that are currently foraging.

In the ABC algorithm, half of the colony consists of employed artificial bees while the other half consists of onlookers. For every food source, there is only one employed bee. In other words, the number of employed bees is equal to the number of food sources around the hive. An employed bee that exhausts its food source becomes a scout.

ABC algorithm simulation for optimization: In the ABC algorithm, the position of a food source i at generation G represents a possible solution to the optimization problem x_i^G, while the nectar amount in a food source corresponds to the quality (fitness fit_i^G) of the associated solution.

Algorithm 1: ABC Algorithm
Requirements: Max Cycles, Colony Size, Limit
Begin
1: Initialize the food sources

$$x_{i,j}^{G=0} = lb_j + rand_j * (ub_j - lb_j) \tag{2.9}$$

where $rand_j$ a random number in [0,1].
2: Evaluate the food sources
3: Cycle = 1
4: while ($Cycle \leq Max\_cycle$) do
5: Produce new solutions using employed bees

$$v_{i,j} = x_{i,j} + \varphi_{i,j} * (x_{i,j} - x_{k,j}) \tag{2.10}$$

where $k \in \{1, 2, .., SN\}$ and $j \in \{1, 2, .., D\}$ are randomly selected indices. Although k is determined randomly, it has to be different from i. φ_{ij} is a random

number between $[-1, 1]$. $v_{i,j}$ is the neighborhood of $x_{i,j}$ in dimension j.
6: Evaluate the new solutions and apply a greedy selection process
7: Calculate the probability values using the fitness values

$$p_i = \frac{fit_i^G}{\sum_{n=1}^{SN} fit_n^G} \tag{2.11}$$

where fit_i^G is the fitness of food source i at generation G.
8: Produce new solutions using onlooker bees

$$v_{i,j} = x_{i,j} + \varphi_{i,j} * (x_{i,j} - x_{k,j}) \tag{2.12}$$

9: Apply a greedy selection process for onlooker bees
10: Determine the abandoned solutions and generate new solutions randomly using scouts

$$x_{ij}^{G=0} = lb_j + rand_j * (ub_j - lb_j) \tag{2.13}$$

where $rand_j$ a random number in $[0, 1]$.
11: Memorize the best solution found so far
12: $Cycle = Cycle + 1$
13: end while
14: return best solution
End

2.7 Particle Swarm Optimization (PSO)

Among the modern meta-heuristic algorithms, a well-known branch is Particle Swarm Optimization (PSO) [5, 40]. PSO is a robust stochastic optimization algorithm which is defined by the behavior of a swarm of particles in a multi-dimensional search space looking for the best solution. It has been developed through simulation of a simplified social system, and has been found to be robust in solving optimization problems. PSO is the method using simple iterative calculations, thus it is easy to create the program source. Therefore, PSO is applicable to wide-ranging optimization problems. Nevertheless, the performance of

the PSO greatly depends on its parameters and it often suffers from the problem of being trapped in the local optimum. It might be difficult to find the global optimal solution when it comes to complex objective functions which have a lot of local optimal solutions. The main problem of PSO is that it prematurely converges to stable point which is not necessary optimum. To resolve this problem, various improvement algorithms are proposed to be a successful in solving a variety of optimal problems [6, 14, 57].

2.7.1 PSO Algorithm

We concerned here with conventional basic model of PSO [40]. In this model, each particle which make up a swarm has information of its position x_i and velocity v_i (where i is the index of the particle) at the present in the search space. Each particle aims at the global optimal solution by updating next velocity making use of the position at the present, based on its best solution has been achieved so far $Lbest_{ij}$ and the best solution of all particles $Gbest_j$ (where $j = [1, 2, \ldots, D]$, D is the dimension of the solution vector), as following equation:

$$v_{ij,k+1} = wv_{ij,k} + c_1 r_1 \left(Lbest_{ij,k} - x_{ij,k}\right) + c_2 r_2 \left(Gbest_{j,k} - x_{ij,k}\right) \ , \tag{2.14}$$

$$x_{ij,k+1} = x_{ij,k} + v_{ij,k+1} \ , \tag{2.15}$$

where w is inertia weight; c_1 and c_2 are cognitive acceleration and social acceleration, respectively; r_1 and r_2 are random numbers uniformly distributed in the range [0.0,1.0].

The position of each particle is updated with (2.15) by the velocity updated in (2.14) as shown in Figure 2.9. After a number of iterations, PSO is going to get the global optimal solution as conclusive *gbest*.

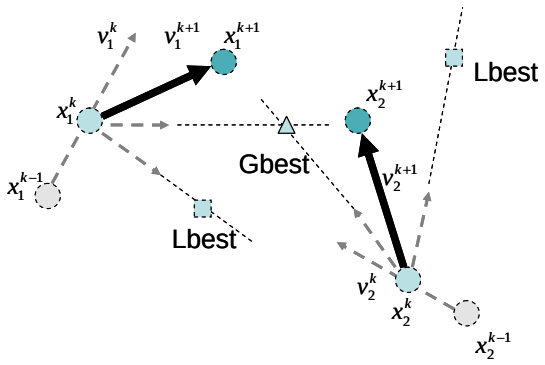

Figure 2.9: Image of PSO algorithm.[40]

2.7.2 Improved PSO algorithm

Given its simple concept and effectiveness, the PSO has become a popular optimizer and has widely been applied in practical problem solving. Meanwhile, much research on performance improvements has been reported including parameter studies, combination with auxiliary operations, and topological structures.

1. Time-varying Inertia Weight

   The concept of *time-varying* has been adapted for improvement of PSO [58]. The inertia weight w in (2.14) linearly decreasing with the iterative generation as below:

   $$w = (w_{\max} - w_{\min}) \left(\frac{iter_{\max} - iter}{iter_{\max}} \right) + w_{\min} \ , \tag{2.16}$$

   where $iter$ is the current iteration number while $iter_{\max}$ is the maximum number of iterations, the maximal and minimal weights $w_{\max}$ and $w_{\min}$ are respectively set 0.9, 0.4 known from experience.

   The concept of diversification and intensification is quite important in PSO algorithm, because it decides the characteristic of the swarm and the search performance. By using (2.16), the particles can be transformed from diversification to intensification by decreasing the inertia weight linearly as the search proceeds.

2. Time-varying Acceleration Coefficients

   The acceleration coefficients c_1 and c_2 are also important parameters in PSO. Both acceleration coefficients are essential to the success of PSO. The idea behind time-varying acceleration coefficients is to enhance the global search in early part of the optimization and to encourage the particles to converge towards the global optima at the end of the search proceeds. With a large cognitive component and small social component at the beginning, particles are allowed to move around the search space instead of moving toward the population best during early stages. On the other hand, a small cognitive component and a large social component allow the particles to converge to the global optima in the latter part of the optimization process. The acceleration coefficients are expressed as:

$$c_1 = (c_{1f} - c_{1i}) \left(\frac{iter_{\max} - iter}{iter_{\max}} \right) + c_{1i} \ , \tag{2.17}$$

$$c_2 = (c_{2f} - c_{2i}) \left(\frac{iter_{\max} - iter}{iter_{\max}} \right) + c_{2i} \ , \tag{2.18}$$

   where c_{1i}, c_{1f}, c_{2i} and c_{2f} are initial and final values of the acceleration coefficient factors respectively. The most effective values are set 2.5 for c_{1i} and c_{2f} and 0.5 for c_{1f} and c_{2i} as in [13].

3. Constriction Factor

   The constriction factor [7] is used to improve the convergence of PSO algorithm and is given by:

$$K = \frac{2}{\left| 2 - \varphi - \sqrt{\varphi^2 - 4\varphi} \right|} \ ; \ \varphi = c_1 + c_2; \ \varphi \geq 4 \ . \tag{2.19}$$

Another active research trend in PSO is hybrid PSO, which combines PSO with other evolutionary paradigms such as Particle Swarm Inspired Evolutionary Algorithm (PS-EA) [60], Hybrid Genetic Algorithm and Particle Swarm Optimization (GA-PSO) [37], and Particle Swarm Ant Colony Optimization (PSACO)

[56] etc. All techniques have also been hybridized with traditional PSO to enhance performance and to prevent the swarm from crowding too closely and to locate as many optimal solutions as possible.

Chapter 3

Improve Seft-Adaptive Control Parameters in Differential Evolution Algorithm

In the chapter 2 we review some metaheuristic algorithm for global optimization. In this chapter, to improve the global search ability and stability of metaheuristic algorithms, we proposed a new improvement of self-adaptive strategy for controlling parameters in differential evolution algorithm (ISADE). The differential evolution (DE) algorithm has been used in many practical cases and has demonstrated good convergence properties. It has only a few control parameters as number of particles (NP), scaling factor (F) and crossover control (CR), which are kept fixed throughout the entire evolutionary process. However, these control parameters are very sensitive to the setting of the control parameters based on their experiments. The value of control parameters depend on the characteristics of each objective function, so we have to tune their value in each problem that mean it will take too long time to perform. We present a new version of the DE algorithm for obtaining self-adaptive control parameter settings that show good performance on numerical benchmark problems and constrained engineering optimization problems.

3.1 Introduction

Differential evolution is an optimization technique originally proposed by R.Storn and K.Price [61]. In DE, new individuals are generated by mutation and crossover operator, which uses the variance within the population to guide the choice of new search points. Although DE is very powerful, it may sometime fall into local optimum and has a slow convergence speed in the last period of iterations. The aim of this work is to improve self-adaptive differential evolution, to do this the three DE's mutation scheme operators are selected as candidates due to their good performance on problems with different characteristics. These three mutation scheme operators are chosen to be applied to individuals in the current population with the same probability. The scaling factor F is calculated by ranking the population and applying formula of sigmoid function depend on the rank number of population size and the crossover control CR is also adaptively changed instead of taking fixed values to deal with different classes of problems. Another critical parameter of DE, the population size NP remains a user-specified variable to tackle problems with different complexity. The results from experiment show that our algorithm with improve self-adaptive control parameter settings is better than or at least comparable to the standard DE algorithm and evolutionary algorithms (EAs) from literature when considering the quality of the solutions obtained and calculation cost. All the algorithms are applied to the some benchmark functions and compared based on some different metrics.

3.2 Review of DE and related work

3.2.1 Formulation of Optimization Problem

The optimization problem is formulated in this section. Design variable vectors (DVs), objective function and range of DVs are defined as follow:

Design variable:

$$x = [x_1, ..., x_D] \tag{3.1}$$

Objective function:

$$f(x) \rightarrow Minimize \tag{3.2}$$

Modified objective function:

$$f^*(x) = f(x) + \gamma P(x) \rightarrow Minimize \tag{3.3}$$

Inequality constraint functions:

$$g_j(x) \leq 0, \quad j = 1, \ldots, m \tag{3.4}$$

Equality constraint functions:

$$h_k(x) = 0, \quad k = 1, \ldots, n \tag{3.5}$$

Range of design variables:

$$x_i^{lb} \leq x_i \leq x_i^{ub} \tag{3.6}$$

where $f(x)$, γ and $f^*(x)$ denote objective function, penalty coefficient and modified objective function, respectively. $x^{lb} = [x_1^{lb}, \ldots, x_D^{lb}]$, $x^{ub} = [x_1^{ub}, \ldots, x_D^{ub}]$, and D denote the lower boundary condition vectors, upper boundary condition vectors, and number of design variable vectors, respectively. m and n are the number of inequality and equality constraints respectively. g_j and h_k are linear or nonlinear real-value functions respectively.

3.2.2 Review of Differential Evolution Algorithm

Differential evolution (DE), proposed by R.Storn and K.Price [61], is a very popular EA. Like other EAs, DE is a population-based stochastic search technique. It uses mutation, crossover and selection operators at each generation to move its population toward the global optimum minimum.

3.2.2.1 Initialization in DE

The initial population was generated uniformly at random in the range lower boundary (LB) and upper boundary (UB).

$$X_{ij}^{G=0} = lb_j + rand_j(0,1) * (ub_j - lb_j) \quad rand_j(0,1) \text{ a random number in [0,1].} \tag{3.7}$$

3.2.2.2 Mutation operation

In this process, DE creates a mutant vector $V_i^G = (V_{i,1}^G, \cdots, V_{i,D}^G)$ for each individual at each generation G, X_i^G (called a target vector) in the current population.

There are several variants of DE, according to [61, 62] we have some mutation schemes as follow:

DE/rand/1:

$$V_{i,j}^G = X_{r_1,j}^G + F * (X_{r_2,j}^G - X_{r_3,j}^G) \tag{3.8}$$

DE/best/1:

$$V_{i,j}^G = X_{best,j}^G + F * (X_{r_1,j}^G - X_{r_2,j}^G) \tag{3.9}$$

DE/current to best/1:

$$V_{i,j}^G = X_{i,j}^G + F * (X_{best,j} - X_{i,j}^G) + F * (X_{r_1,j}^G - X_{r_2,j}^G) \tag{3.10}$$

DE/rand/2:

$$V_{i,j}^G = X_{r_1,j}^G + F * (X_{r_2,j}^G - X_{r_3,j}^G) + F * (X_{r_4,j}^G - X_{r_5,j}^G) \tag{3.11}$$

DE/best/2:

$$V_{i,j}^G = X_{best,j}^G + F * (X_{r_1,j}^G - X_{r_2,j}^G) + F * (X_{r_3,j}^G - X_{r_4,j}^G) \tag{3.12}$$

DE/rand to best/1:

$$V_{i,j}^G = X_{r_1,j}^G + F * (X_{best,j} - X_{r_1,j}^G) + F * (X_{r_2,j}^G - X_{r_3,j}^G) \tag{3.13}$$

where r_1, r_2,r_3, r_4 and r_5 are distinct integers that randomly selected from the range $[1, NP]$ and are also different from i. The parameter F is called the scaling factor that amplifies the difference vectors. X_{best} is the best individual in the current population.

3.2.2.3 Crossover operation

After mutation process, DE performs a binomial crossover operator on X_i^G and V_i^G to generate a trial vector $U_i^G = (U_{i,1}^G, \cdots, U_{i,D}^G)$ for each particle i as shown in (Eq.3.14).

$$U_i^G = \begin{cases} V_{i,j}^G & \text{if } rand_j(0,1) \leq CR \text{ or } j = j_{rand} \\ X_{i,j}^G & \text{otherwise.} \end{cases} \tag{3.14}$$

where $i = 1, \cdots, NP$, $j = 1, \cdots, D$, j_{rand} is a randomly chosen integer in $[1, D]$, $rand_j(0, 1)$ is a uniformly distributed random number between 0 and 1 generated for each j and $CR \in [0, 1]$ is called the crossover control parameter. Due to the use of j_{rand}, the trial vector U_i^G differs from target vector X_i^G.

3.2.2.4 Selection operation

The selection operator is performed to select the better one between the target vector X_i^G and the trial vector U_i^G to enter the next generation.

$$X_i^{G+1} = \begin{cases} U_i^G & \text{if } f(U_i^G) \leq f(X_i^G) \\ X_i^G & \text{otherwise.} \end{cases} \tag{3.15}$$

where $i = 1, \cdots, NP$, X_i^{G+1} is target vector in the next population.

3.2.3 Related work of Differential Evolution Algorithm

This section reviews some papers that compared the different extension of DE with the original DE. After that, we concentrate on papers that deal with parameter control in DE.

There have been many research works on controlling search parameters of DE that are NP, F and CR.

R.Storn and K.Price [61] argued that these three control parameters are not difficult to set for obtaining good performance. They suggested that NP should be between $5D$ and $10D$, F should be 0.5 as a good initial choice and the value

Algorithm 1 The DE pseudo-code

```
Require: NP, CR and F Parameters.
INITIALIZE DE randomly creates population in (Eq.3.7);
EVALUATE Calculate fitness of each individuals ;
while (TERMINATION CONDITION) do
    Mutation DE creates a mutation vector V in (Eq.3.8 to Eq.3.13);
    Crossover DE creates a trial vector U in (Eq.3.14) ;
    Evaluation Calculate fitness value of offspring;
    Selection select the better one between the X_i^G and the U_i^G for next
    generation in (Eq.3.15);
    Memorize Best solution found so far;
end while
```

of F smaller than 0.4 or larger than 1.0 will lead to performance degradation and CR can be set to 0.1 or 0.9.

Omar S.Soliman and Lam T.Bui at [59], the author introduced a self-adaptive approach to DE parameters using a variable step length generated by a Gaussian distribution; also, the mutation amplification and crossover parameter were introduced. These parameters are evolved during the optimization process.

A.K.Qin and P.N.Suganthan [3] proposed the new choice of learning strategy SaDE and the two control parameters F and CR do not require predetermining. During evolution, suitable learning strategy and parameter are applied. Here, author proposed learning strategy adaptation is to probabilistically select one out of several available learning strategies and apply to the current population. The reason for author's choice is that these two strategies have been commonly used in many DE literature and reported to perform well on problems with distinct characteristics. Among them, *"rand/1/bin"* strategy usually demonstrates good diversity while the *"current to best/2/bin"* strategy shows good convergence property, two candidate strategies, assuming that the probability of applying strategy "rand/l/bin" to each individual in the current population is p_1 , the probability of applying another strategy should be $P_2 = 1 - p_1$. After specified number of generations called the "learning period".Then, the probability of p_1 is updated. The author considered allowing F to take different random values in the range

$(0, 2]$ with normal distributions of mean 0.5 and standard deviation 0.3 for different individuals in the current population. For CR author assumed CR normally distributed in a range of normal distribution of CR (CRm) and standard deviation 0.1. The CR values associated with trial vectors successfully entering the next generation are recorded. After a specified number of generations CR has been changed for several times under the same normal distribution with center CRm and standard deviation 0.1, and author recalculated the CRm according to all the recorded CR values corresponding to successful trial vectors during this period.

J.Liu and J.Lampinen [44] present an algorithm based on the Fuzzy Logic Control (FLC) in which the step-length was controlled using a single FLC. Its two inputs were: linearly depressed parameter vector change and function value change over the whole population members between the current generation and the last generation.

J.Teo [35] proposed an attempt to dynamic self-adaptive populations in differential evolution, in addition to self-adapting crossover and mutation rates, they showed that DE with self-adaptive populations produced highly competitive results compared to a conventional DE algorithm with static populations.

J.Brest [29] presented another variant of DE algorithms jDE, which uses different self-adaptive mechanisms applied on the control parameters: The step length F and crossover rate CR are produce factors F and CR in a new parent vector.

$$F_i^{G+1} = \begin{cases} F_l + rand_1 * F_u & \text{if } rand_2 \leq \tau_1 \\ F_i^G & \text{otherwise.} \end{cases} \tag{3.16}$$

$$CR_i^{G+1} = \begin{cases} rand_3 & \text{if } rand_4 \leq \tau_2 \\ CR_i^G & \text{otherwise.} \end{cases} \tag{3.17}$$

where $rand_1$, $rand_2$, $rand_3$, $rand_4$ are uniform random values $\in [0, 1]$. τ_1 and τ_2 represent probabilities to adjust factors F and CR, respectively. Author set $\tau_1 = \tau_2 = 0.1$. Because $F_l = 0.1$ and $F_u = 0.9$, the new takes a value form $[0.1, 0.9]$ in a random manner. The new CR takes a value from $[0, 1]$. F_i^{G+1} and CR_i^{G+1} are obtained before the mutation process.

Through reviewing related work, we understood that it is difficult to select DE learning strategies in the mutation operator and DE control parameters. To overcome this drawback we proposed the Improvement of Self-Adapting control parameters in Differential Evolution (ISADE) - a new version of DE in this research. The detail of ISADE is presented in the next section.

3.3 Improvement of Self-Adapting Control Parameters in Differential Evolution

From literature review of DE, the DE algorithm can work outstanding in comparison others evolutionary algorithm. How ever to achieve good performance on a specific problem by using the original DE algorithm, we need to try all available (usually 6 mutation schemes mentioned in section 3.2.2.2) learning strategies in the mutation operator and fine-tune the corresponding critical control parameters CR, F and NP. Through reviewing related work, we know that the performance of the original DE algorithm is highly dependent on the strategies and parameter settings. Although we may find the most suitable strategy and the corresponding control parameters for a specific problem, it may require a huge amount of computation time. Also, during different evolution stages, different strategies and corresponding parameter settings with different global and local search capability might be preferred. Therefore, to overcome this drawback, we attempt to develop a new version of DE algorithm that can automatically adapt the learning strategies and the parameters settings during evolution. The main ideas of the ISADE algorithm are summarized below.

3.3.1 Adaptive selection learning strategies in the mutation operator

ISADE probabilistically selects one out of several available learning strategies in the mutation operator for each individual in the current population. Hence, we should have several candidate learning strategies available to be chosen and

also we need to develop a procedure to determine the probability of applying each learning strategy. In this research, we select three learning strategies in the mutation operator as candidates: "DE/best/1/bin", "DE/best/2/bin" and "DE/rand to best/1/bin". The reason for author's choice is that these three strategies have been commonly used in many DE literature and reported to perform well on problems with distinct characteristics [61, 62]. Among them, *"DE/rand to best/1"* strategy usually demonstrates good diversity (explore ability) while the *"DE/best/1"* and *"DE/best/2"* strategy shows good convergence property (exploitation ability, which we also observe in our trial experiments. Since here we have three candidate strategies, the probability of applying strategy to each particle in the current population is p_i which are same value $p_1 = p_2 = p_3 = 1/3$. With this learning strategies in the mutation operator, the procedure can gradually evolve the most suitable learning strategy at different learning stages for the problem under consideration. After specified number of generations called the "learning period"

DE/rand to best/1:

$$V_{i,j}^{G} = X_{r_1,j}^{G} + F * (X_{best,j} - X_{r_1,j}^{G}) + F * (X_{r_2,j}^{G} - X_{r_3,j}^{G}) \tag{3.18}$$

DE/best/1:

$$V_{i,j}^{G} = X_{best,j}^{G} + F * (X_{r_1,j}^{G} - X_{r_2,j}^{G}) \tag{3.19}$$

DE/best/2:

$$V_{i,j}^{G} = X_{best,j}^{G} + F * (X_{r_1,j}^{G} - X_{r_2,j}^{G}) + F * (X_{r_3,j}^{G} - X_{r_4,j}^{G}) \tag{3.20}$$

3.3.2 Adaptive scaling factor F

As mentioned in section 3.3 before applying original DE algorithm for optimization problem we have to tune scaling factor F. The scaling factor F is sensitive and depend on each of problem, for each of optimization problem we have tune it's value that mean it require huge amount of computation time. Therefore, to overcome of this drawback we try to automatically get scaling factor F value. to do this matter let consider the situation in the Fig. 3.1. In the multi-point search

of the DE, we have many particles move from their current points to new search points in the design space of design variables. For example , as shown in Fig. 3.1, we have three particles, the first particle A has hight fitness value and near the global minimize pint and it requires a slight change to the values of the design variables to obtain the global optimum solution. On the other hand, particle B cannot reach a global optimum solution without a significant change, and in addition, particle C has landed in a local optimum solution. Such a situation, in which the good individual and the low individual are intermingled, can generally occur at any time in this search process. Therefore, we have to recognize each individual's situation and propose a suitable design variables generation process for each individual's situation in the design space.

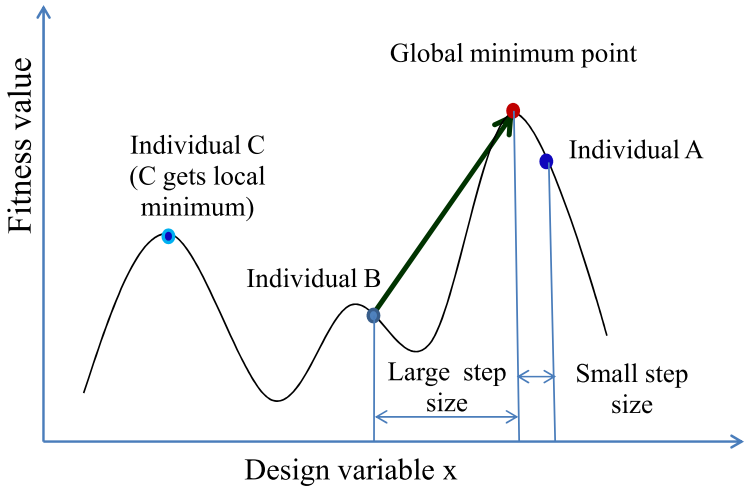

Figure 3.1: Example of individual situations.

From the above analysis we give the following conclusions:

1. *In the multi-point search of population, the point which has high fitness value will near global point and requires a slight change to the values of the design variables to obtain the global optimum solution.*
2. *On the other hand, the point which low fitness value will far from global point and cannot reach a global optimum solution without a significant change.*

In the DE algorithm, the distance for a search point can be changed by controlling the F factor for determining the neighborhood range. To do this, S.Tooyama

and H.Hasegawa [65] proposed APGA/VNC approach in which author used sigmoid function to control neighborhood parameter shown as in 3.2. The formula for F by sigmoid function as follows.

$$F_i = \frac{1}{1 + exp\left(\alpha * \frac{i - \frac{NP}{2}}{NP}\right)} \tag{3.21}$$

where α, i denote the gain of the sigmoid function, particle of i^{th} in NP, respectively.

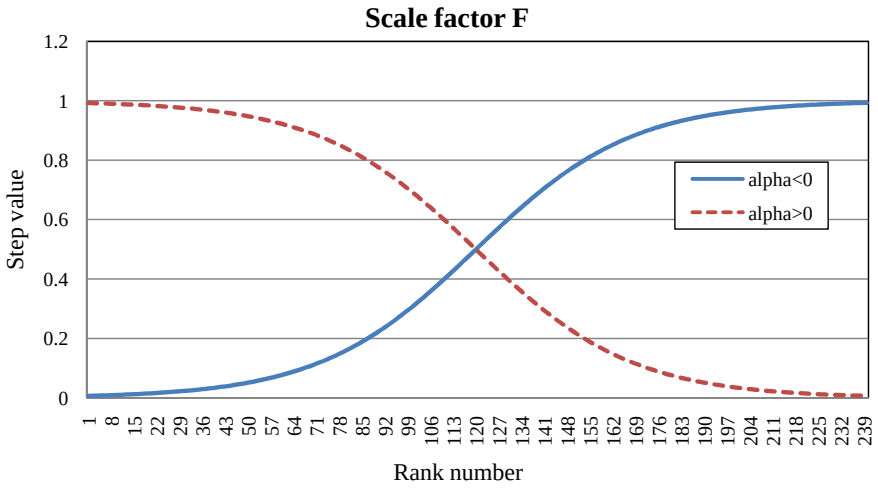

Figure 3.2: Suggested to calculate F value.

The gait of F chart depends on the sign and gain of α Fig. 3.2. When particle at good fitness (high fitness) same as particle A in Fig. 3.1 will have small step size of F factor and otherwise in the case of $\alpha < 0$ (the continuous line in Fig. 3.1). From this view, the ISADE method automatically adapts F factor to obtain design variable generation accuracy for each individual's situation and particle's fitness. As a result, we believe that it will steadily provide a global optimum solution and reduce the calculation cost.

In this paper, before caculate the scale factor F_i in Eq.3.21. we will rank all the particles by estimating their fitness. A ranked particle is labeled with this rank number and assigned F that corresponds with this number.

For better performance of ISADE it is need that the scale factor F should be high in the beginning to have much exploration and after certain generation F is need to be small for proper exploitation. To implement this, we have new approach to calculate the the scale factor F as follow:

$$F_{iter}^{mean} = F_{min} + (F_{max} - Fmin)\left(\frac{iter_{max} - iter}{iter_{max}}\right)^{n_{iter}} \tag{3.22}$$

where F_{max}, F_{min}, $iter$, $iter_{max}$ and n_{iter} denote the lower boundary condition of the F, and the upper boundary condition of the F, maximum generation, current generation and nonlinear modulation index, respectively. From our experiment we assign $F_{min} = 0.15$, and $F_{max} = 1.55$.

To control the F_{iter}^{mean}, we have varied the nonlinear modulation index n_{iter} with generation as follows:

$$n_{iter} = n_{min} + (n_{max} - n_{min})\left(\frac{iter}{iter_{max}}\right) \tag{3.23}$$

where n_{max} and n_{min} are typically chosen in the range $(0, 15]$. After a number of experiments on the values of n_{max} and n_{min}, we have found that the best choice for them is 0.2 and 6.0. The gait of F_{iter}^{mean} chart depends on the iteration number and the nonlinear modulation index n_{iter} is shown in Fig. 3.3.

We introduced a novel approach of scale factor F_i of the each particles with their fitness values in (Eq.3.21). Therefore in one generation the value of F_i^{iter} $(i = 1, \cdots, NP)$ are not the same for all particles in the population rather it is made to vary for all particles in each generation. Consider F_{iter}^{mean} of (Eq.3.22) as an average value that we assign to each generation and the final value of scale factor for each particle in each generation is calculated as follow:

$$F_{iter}^{i} = \frac{F_i + F_{iter}^{mean}}{2} \tag{3.24}$$

where $iter = 1, \ldots, iter_{max}$ and $i = 1, \ldots, NP$

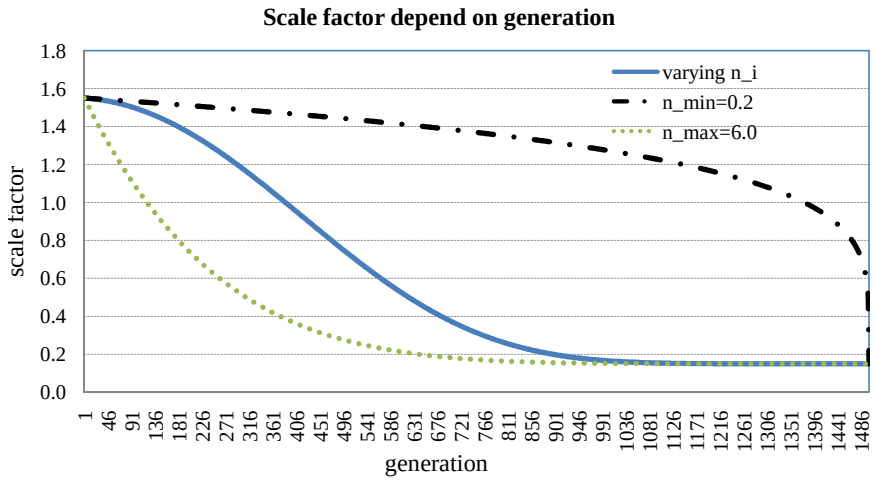

Figure 3.3: The scale factor depend on generation.

3.3.3 Adaptive crossover control parameter CR

In the crossover operation of DE section 3.2.2.3 the crossover control parameter CR decide the mutant vector V_{ij}^G become a trial vector U_{ij}^G otherwise target vector X_{ij}^G. same as the scaling factor F before running DE algorithm we have to tune it's value. In this section we will try to automatically get it's value.

There are many research focus to this problem [3, 29, 53, 59, 61]. base on G.Reynoso-Meza [53] suggested to have a success if a child substitutes its parent in the next generation.The minimum, maximum and medium value on such set of success is used for this purpose.

- Be able to detect a *separable problem or independent problem*, choosing a crossover control parameter with low values for CR. Fig. 3.4

- Be able to detect *non-separable problem or dependent problem*, choosing a crossover control parameter with high values for CR. Fig. 3.4

In this way, the algorithm will be able to detect if high values of CR are useful and furthermore, if a rotationally invariant crossover is required. A minimum base for CR around its median value is incorporated to avoid stagnation around a single value, Fig. 3.4 shows this principle, so we propose the ideas behind this adaptive mechanism for the crossover:

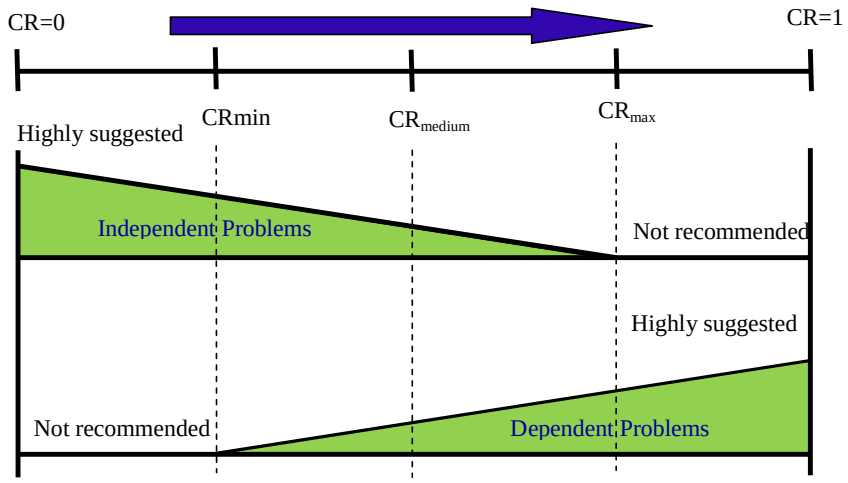

Figure 3.4: Suggested to calculate CR values.

The control parameter CR is adapted as follows:

$$CR_i^{G+1} = \begin{cases} rand_2 & \text{if } rand_1 \leq \tau \\ CR_i^G & \text{otherwise.} \end{cases} \tag{3.25}$$

where: $rand_1$ and $rand_2$ are uniform random values $\in [0, 1]$, τ represents probabilities to adjust CR, same as [30] we assign $\tau = 0.10$.

After that we adjust CR as follows:

$$CR_i^{G+1} = \begin{cases} CR_{min} & \text{if } CR_{min} \leq CR_i^{G+1} \leq CR_{medium} \\ CR_{max} & \text{if } CR_{medium} \leq CR_i^{G+1} \leq CR_{max} \end{cases}. \tag{3.26}$$

where: CR_{min}, CR_{medium} and CR_{max} denote the low value, median value and high value of crossover parameter respectively. From our experiment in many trials, we assign $CR_{min} = 0.05$, $CR_{medium} = 0.50$ and $CR_{max} = 0.95$.

The purpose of our approach is that user does not need to tune the good values for F and CR, which are problem dependent. The rules for improve self-adapting control parameters are quite simple, therefore the new version of the DE algorithm does not increase the time complexity in comparison to the original DE algorithm.

3.3.4 ISADE algorithm pseudo-code

Algorithm 2 The ISADE pseudo-code

Require only NP parameter;
INITIALIZE DE randomly creates population in (Eq.3.7);
EVALUATE Calculate fitness of each individuals and rank population in ther decent fitness ;
while (TERMINATION CONDITION) **do**
    Adaptive scaling factor F in (Eq.3.21) to (Eq.3.24);
    Adaptive Crossover factor CR in (Eq.3.25) to (Eq.3.26);
    Mutation DE creates a mutation vector V in (Eq.3.8 to Eq.3.13);
    Crossover DE creates a trial vector U in (Eq.3.14) ;
    EVALUATE Calculate fitness of each individuals and rank population in ther decent fitness ;
    Selection select the better one between the X_i^G and the U_i^G for next generation in (Eq.3.15);
    Memorize Best solution found so far;
end while

3.4 Numerical Experiments

In this section, the first experiment is to be test for turning the α parameter ISADE, after that we will test the robustness of the ISADE method compare with some reference methods as: jDE [30], PSO [39] and SaDE [3], the last experiment, we will apply ISADE to some constrained engineering optimizations. These experiments are performed 20 trials for every function. .

3.4.1 Benchmark Tests

To estimate the stability and convergence to the optimal solution of ISADE, We will use 9 well-known benchmark test functions with 30 dimensions such as Sphere (Sp), Rosenbrock (Ro), Ridge (Ri), Griewank (Gr), Rastrigin (Ra), Ackley (Ac),

Levy (Le), Schawefel (Sc) and Alpine (Al) function. These functions are given as follows:

Sphere:

$$f_1 = \sum_{i=1}^{n} \{x_i^2\} \tag{3.27}$$

Rosenbrock:

$$f_2 = \sum_{i=1}^{n} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \tag{3.28}$$

Ridge:

$$f_3 = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{3.29}$$

Griewank:

$$f_4 = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{3.30}$$

Rastrigin:

$$f_5 = 10n + \sum_{i=1}^{n} \{x_i^2 - 10cos(2\pi x_i)\} \tag{3.31}$$

Ackley:

$$f_6 = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e \tag{3.32}$$

Levy:

$$f_7 = sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)\left(1 + sin^2(3\pi x_{i+1})\right) + (x_n - 1)\left(1 + sin^2(2\pi x_n)\right) \tag{3.33}$$

Schawefel:

$$f_8 = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i| \tag{3.34}$$

Alpine:

$$f_9 = \sum_{i=1}^{n} |x_i sin(x_i) + 0.1x_i| \tag{3.35}$$

Table 3.1 shows the characteristics, the terms dependent or independent problem, multi-peak, steep denote the dependence relation of the variables, presence of multi-peak and level of steepness, respectively.

Table 3.1: Characteristics of Benchmark Functions.

Func	Dependent	Multi-peak	Steep	Design range	Global opt
Sp	No	no	Average	$-5.12 \leq x \leq 5.12$	$f(0) = 0$
Ro	Yes	No	Big	$-2.048 \leq x \leq 2.048$	$f(1) = 0$
Ri	Yes	No	Average	$-51.2 \leq x \leq 51.2$	$f(0) = 0$
Gr	Yes	Yes	Small	$-600.0 \leq x \leq 600.0$	$f(0) = 0$
Ra	No	Yes	Average	$-5.12 \leq x \leq 5.12$	$f(0) = 0$
Ac	No	Yes	Average	$-5.12 \leq x \leq 5.12$	$f(0) = 0$
Le	No	Yes	Average	$-10.0 \leq x \leq 10.0$	$f(1) = 0$
Sc	No	No	Average	$-10.0 \leq x \leq 10.0$	$f(0) = 0$
AL	Yes	Yes	Average	$-10.0 \leq x \leq 10.0$	$f(0) = 0$

3.4.2 Test to get best value of α in ISADE

As mention above, we know that the gait of sigmoid function is depend on the sign and value of α, Fig. 3.2 shows the relationship of F and $Rank$ depend on the α.

In this section, we test to get best value of α. $NP = 8 * D$, maximum iteration $iter\_max = 3000$, accurate $\varepsilon = 10^{-6}$, $\tau = 0.1$ and alpha value $\alpha = -20, -19, \ldots, 19, 20$.

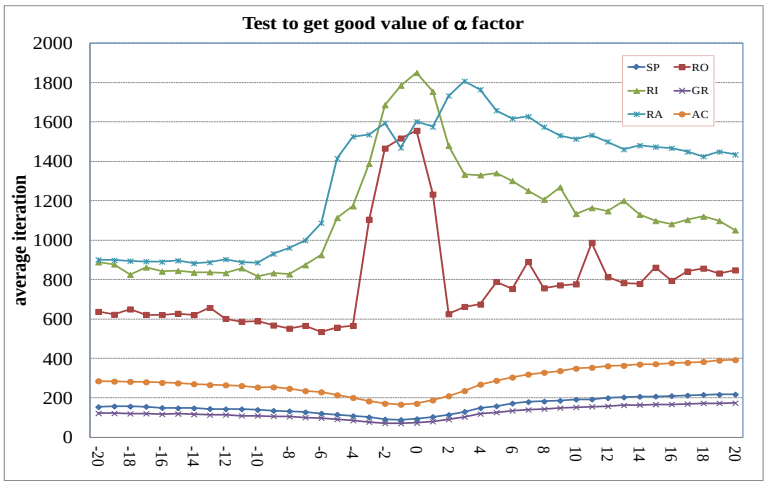

Figure 3.5: Result of test to get good value of α.

From the result in Fig. 3.5, the solutions of all the benchmark functions with 30 dimensions reach their global optimum solutions with accurate $\varepsilon = 10^{-6}$. However, the result of $\alpha < 0$ is better than that of $\alpha > 0$. The Sp, Ac, and Gr function are good with α from -10.0 to 10.0 but with Ro, Ri and Ra function is not good at that value, from -10.0 to 20.0 maybe they are not change when we change α value, for best value of all function we will choose $\alpha = -10$ for next test.

3.4.3 Test to robust of Algorithm

3.4.3.1 ISADE and some approaches are compared in this test with same accurate $\varepsilon = 10^{-6}$

Population size was $NP = 8 * D$, and accurate $\varepsilon = 10^{-6}$ compare the iteration, $\tau = 0.1$, at which the optimum is satisfy. The result of this test is shown in table 3.2. We present the average of generations and success ratio for the benchmark functions. All the readings given are the averages over 20 independent runs per function.

Table 3.2: Average of generation and the success ratio

Function	**PSO**		**jDE**		**SaDE**		**ISADE**	
	ave_iter	SR	ave_iter	SR	ave_iter	SR	ave_iter	SR
Sp	196.55	100%	568.90	100%	238.75	100%	**107.90**	100%
Ro	–	0%	5663.15	100%	1084.06	85%	**781.40**	100%
Ri	1517.15	80%	8325.70	100%	766.95	95%	**494.03**	100%
Gr	155.30	100%	457.45	100%	181.25	100%	**128.90**	100%
Ra	–	0%	3563.65	100%	1200.8	100%	**1164.75**	100%
Ac	390.45	100%	973.15	100%	414.50	100%	**305.70**	100%
Le	298.35	100%	655.40	100%	**264.00**	100%	357.30	95%
Sc	592.85	85%	996.50	100%	**473.10**	100%	**449.70**	100%
Al	847 .85	20%	–	0%	1898.30	100%	**966.30**	100%

As in table 3.2, ISADE method compare with some reference methods as: jDE [29], PSO [39] and SaDE [3]. We wish to mention here that ISADE is able obtain the optimal solutions for almost benchmark test functions except Le function, with the success ratio equal 100% . The ISADE also gets global optimum at less iteration than that of reference in all benchmark functions, so ISADE could certainly achieve optimal solution with low calculation cost. With four functions Ro, Ri, Ra and Al the new approach ISADE is very strong robust. The convergence of the optimal solution could be improved more significantly in ISADE than that in EAs for the same accurate.

3.4.3.2 Test with maximum iteration compares the mean of global minimum and (Std) standard deviation

In this experiment, we did test with $N = 8 * D$, $D = 30$ dimensions, $\tau = 0.1$, process will be stop at maximum generation. The mean and standard deviation best fitness of these 20 independent runs has been reported. The result of this test is shown in table 3.3.

The comparison results are listed in table 3.3, all of benchmark functions are tested with maximum iteration. From the results, it can be seen that the average of best fitness value and standard deviation of ISADE is better than that of reference jDE [29], PSO [39] and SaDE [3].

Overall, ISADE was capable of attaining robustness, high quality, low calculation outstanding efficient performance on many benchmark problems. We confirmed satisfactory performance through various benchmark tests.

3.4.4 Solve some real constrained engineering design optimization problems

In this section, we will apply HISADE to solve some real constrained engineering design optimization problems. A set of 4 engineering design optimization problems was chosen to evaluate the performance of our proposed algorithm.

Table 3.3: ($Mean$) Average of global minimum and (std) the standard deviation

F	$iter_{max}$	**PSO**		**jDE**		**SaDE**		**ISADE**	
		Mean	std	Mean	std	Mean	std	Mean	std
Sp	500	$2.83E-12$	$8.24E-12$	$1.14E-05$	$2.77E-06$	$1.32E-11$	$5.77E-11$	**4.84E-36**	$1.58E-35$
Ro	1500	$2.08E+01$	$2.72E+00$	$1.77E+01$	$3.10E-01$	$4.81E-18$	$1.94E-17$	**4.45E-28**	$1.27E-27$
Ri	1500	$6.89E-05$	$5.99E-05$	$1.15E+02$	$3.19E+01$	$2.06E-14$	$6.18E-14$	**3.06E-26**	$1.19E-25$
Gr	500	$1.61E-14$	$3.26E-14$	$2.67E-07$	7.53E-08	**2.22E-15**	$8.00E-15$	**3.61E-16**	$1.21E-16$
Ra	1500	$3.16E+01$	$1.04E+01$	$3.29E+01$	$3.29E+00$	$8.28E-12$	$2.74E-11$	**1.32E-14**	$5.22E-15$
Ac	500	$4.12E-07$	$6.67E-07$	$2.19E-03$	$2.55E-04$	$1.49E-07$	$2.63E-07$	**4.09E-14**	$1.03E-14$
Le	500	$6.12E-11$	$1.56E-10$	$1.75E-04$	$4.84E-05$	$4.04E-10$	$1.40E-09$	**2.21E-31**	$2.56E-31$
Sc	500	$6.69E-04$	$1.43E-03$	$7.95E-03$	$1.14E-03$	**1.15E-06**	$2.08E-06$	**1.24E-06**	$2.39E-06$
Al	1500	$5.38E-12$	$4.23E-12$	$5.25E-03$	$6.97E-04$	$3.38E-05$	$3.96E-05$	**1.09E-09**	$3.43E-09$

We performed 20 independent runs per problem. As in chapter 3, we used the following parameters: $NP = 8D$, $\tau = 0.1$.

3.4.4.1 E01: Welded beam design optimization problem

The problem is to design a welded beam for minimum cost, subject to some constraints [42]. Fig. 3.6 shows the welded beam structure which consists of a beam A and the weld required to hold it to member B. The objective is to find the minimum fabrication cost, considering four design variables: x_1, x_2, x_3, x_4 and constraints of shear stress τ, bending stress in the beam σ, buckling load on the bar P_c , and end deflection on the beam δ. The optimization model is summarized in the next equation:

Minimize:

$$f(x) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2)$$

subject to:

$$g_1(x) = \tau(x) - 13,600 \leq 0 \;\; ; \quad g_2(x) = \sigma(x) - 30,000 \leq 0$$

$$g_3(x) = x_1 - x_4 \leq 0 \;\; ; \quad g_4(x) = 0.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \leq 0$$

$$g_5(x) = 0.125 - x_1 \leq 0 \;\; ; \quad g_6(x) = \delta(x) - 0.25 \leq 0 \;\; ; \quad g_7(x) = 6,000 - Pc(x) \leq 0$$

With

$$\tau(x) = \sqrt{(\tau')^2 + (2\tau'\tau'')\,\frac{x_2}{2R} + (\tau'')^2} \;\; ; \quad \tau' = \frac{6,000}{\sqrt{2}x_1x_2} \;\; ; \quad \tau'' = \frac{MR}{J}$$

$$M = 6,000\left(14.0 + \frac{x_2}{2}\right) \;\; ; \quad R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} \;\; ; \quad J = 2\left\{x1x2\sqrt{2}\left[\frac{x_2^2}{4.0} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\}$$

$$\sigma(x) = \frac{504,000}{x_4x_3^2} \;\; ; \quad \delta(x) = \frac{65,856,000}{(30\times10^6)x_4x_3^3}$$

$$Pc(x) = \frac{4.013(30\times10^6)\sqrt{\frac{x_3^2x_4^6}{36.0}}}{196.0}\left(1 - \frac{x_3\sqrt{\frac{30\times10^6}{4(12\times10^6)}}}{28.0}\right)$$

With $0.1 \leq x_1, x_4 \leq 2.0$, and $0.1 \leq x_2, x_3 \leq 10.0$.

Best solution: $x^* = (0.205730, 3.470489, 9.036624, 0.205729)$

Where $f(x^*) = 1.724852$

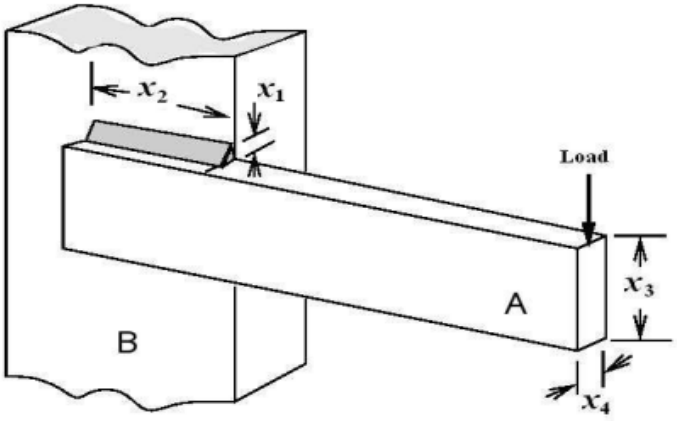

Figure 3.6: Welded Beam.

3.4.4.2 E02: Pressure vessel design optimization problem

A compressed air storage tank with a working pressure of 3,000 psi and a minimum volume of 750 ft^3 . A cylindrical vessel is capped at both ends by hemispherical heads (see Fig. 3.7). Using rolled steel plate, the shell is made in two halves that are joined by toe longitudinal welds to form a cylinder. The objective is minimize the total cost, including the cost of the materials forming the welding [16]. The design variables are: thickness x_1, thickness of the head x_2, the inner radius x_3, and the length of the cylindrical section of the vessel x_4. The variables x_1 and x_2 are discrete values which are integer multiples of 0.0625 inch. Then, The mathematical formulation of this problem is:

Minimize:

$$f(x) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$$

subject to:

$$g_1(x) = -x_1 + 0.0193x_3 \leq 0; \qquad g_2(x) = -x_2 + 0.00954x_3 \leq 0$$

$$g_3(x) = -\pi x_3^2 x_4 - \tfrac{4}{3}\pi x_3^3 + 1,296,000.0 \leq 0; \qquad g_4(x) = x_4 - 240.0 \leq 0$$

With $1 \times 0.0625 \leq x_1, x_2 \leq 99 \times 0.0625$, $10.0 \leq x_3$, and $x_4 \leq 200.0$.

Best solution: $x^* = (0.8125, 0.4375, 42.098446, 176.636596)$

Where $f(x^*) = 6,059.714335$.

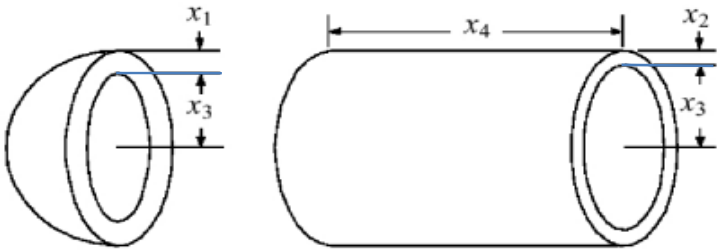

Figure 3.7: Pressure Vessel.

3.4.4.3 E03: Speed reducer design optimization problem

The design of the speed reducer [34] shown in Fig. 3.8, is considered with the face width x_1, module of teeth x_2, number of teeth on pinion x_3, length of the first shaft between bearings x_4, length of the second shaft between bearings x_5, diameter of the first shaft x_6, and diameter of the first shaft x_7 (all variables continuous except x_3 that is integer). The weight of the speed reducer is to be minimized subject to constraints on bending stress of the gear teeth, surface stress, transverse deflections of the shafts and stresses in the shaft. The mathematical formulation of this problem is:

Minimize:

$$\begin{aligned} f(x) \;=\; & 0.7854 x_1 x_2^2 (3.3333 x_3^2 + 14.9334 x_3 - 43.0934) - 1.508 x_1 (x_6^2 + x_7^2) \\ & + 7.4777 (x_6^3 + x_7^3) + 0.7854 (x_4 x_6^2 + x_5 x_7^2) \end{aligned}$$

subject to:

$$g_1(x) = \frac{27.0}{x_1 x_2^2 x_3} - 1.0 \leq 0 \quad ; \quad g_2(x) = \frac{397.5}{x_1 x_2^2 x_3^2} - 1.0 \leq 0$$

$$g_3(x) = \frac{1.93 x_4^3}{x_2 x_3 x_6^2} - 1.0 \leq 0 \quad ; \quad g_4(x) = \frac{1.93 x_5^3}{x_2 x_3 x_7^4} - 1.0 \leq 0$$

$$g_5(x) = \frac{1.0}{110.0 x_6^3} \sqrt{\left(\frac{745.0 x_4}{x_2 x_3}\right)^2 + 16.9 \times 10^6} - 1.0 \leq 0$$

$$g_6(x) = \frac{1.0}{85.0 x_7^3} \sqrt{\left(\frac{745.0 x_5}{x_2 x_3}\right)^2 + 157.5 \times 10^6} - 1.0 \leq 0$$

$$g_7(x) = \frac{x_2 x_3}{40.0} - 1.0 \leq 0 \quad ; \quad g_8(x) = \frac{5.0 x_2}{x_1} - 1.0 \leq 0 \quad ; \quad g_9(x) = \frac{x_1}{12.0 x_2} - 1.0 \leq 0$$

$$g_{10}(x) = \frac{1.5 x_6 + 1.9}{x_4} - 1.0 \leq 0 \quad ; \quad g_{11}(x) = \frac{1.1 x_7 + 1.9}{x_5} - 1.0 \leq 0$$

With $2.6 \leq x_1 \leq 3.6$, $0.7 \leq x_2 \leq 0.8$, $17 \leq x_3 \leq 28$, $7.3 \leq x_4 \leq 8.3$, $7.8 \leq x_5 \leq 8.3$, $2.9 \leq x_6 \leq 3.9$, and $5.0 \leq x_7 \leq 5.5$,

Best solution: $x^* = (3.500000, 0.7, 17, 7.300000, 7.800000, 3.350214, 5.286683)$

Where $f(x^*) = 2,996.348165$.

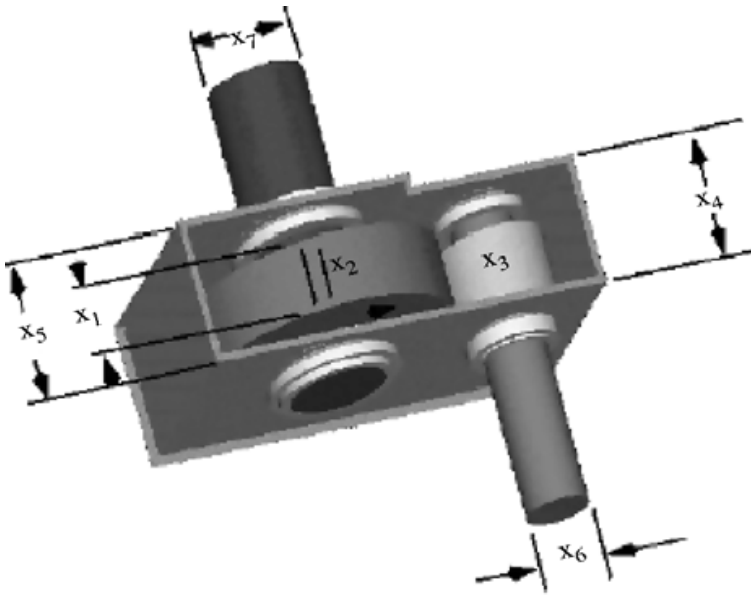

Figure 3.8: Speed Reducer.

3.4.4.4 E04: Tension/compression spring design optimization problem

This problem[2, 32] minimizes the weight of a tension/compression spring Fig. 3.9, subject to constraints of minimum deflections, shear stress, surge frequency, limits on outside diameter and on design variables. There are three design variables: the wire diameter x_1, the mean coil diameter x_2, and the number of active coils x_3. The mathematical formulation of this problem is:

Minimize:

$$f(x) = (x_3 + 2)x_2x_1^2$$

subject to:

$$g_1(x) = 1.0 - \frac{x_2^3 x_3}{7{,}1785x_1^4} \leq 0; \qquad g_2(x) = \frac{4.0x_2^2 - x_1x_2}{12{,}566(x_2x_1^3 - x_1^4)} + \frac{1.0}{5{,}108x_1^2} - 1.0 \leq 0$$

$$g_3(x) = 1.0 - \frac{140.45x_1}{x_2^2x_3} \leq 0; \qquad g_4(x) = \frac{x_1 + x_2}{1.5} - 1.0 \leq 0$$

With $0.05 \leq x_1 \leq 2.0$, $0.25 \leq x_2 \leq 1.3$, and $2.0 \leq x_3 \leq 15.0$.

Best solution: $x^* = (0.051690, 0.356750, 11.287126)$

Where $f(x^*) = 0.012665$.

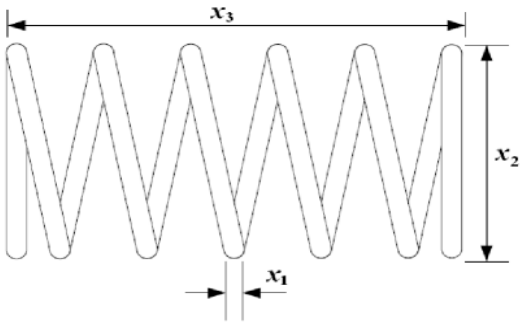

Figure 3.9: Tension/Compression Spring.

Table 3.4: Result of applying ISADE for E01 (Welded beam) problem.

Solutions		Constraints	
x_1	0.205730	$g_1(x)$	-4.60E-07
x_2	3.470489	$g_2(x)$	-1.20E-06
x_3	9.036624	$g_3(x)$	-3.97E-12
x_4	0.205730	$g_4(x)$	-3.97E-12
f(x)	**1.724852**	$g_5(x)$	-3.97E-12
FE	8000	$g_6(x)$	-3.97E-12
		$g_7(x)$	-3.43E+00

Table 3.5: Result of applying ISADE for E02 (Pressure vessel) problem.

Solutions		Constraints	
x_1	0.812500	$g_1(x)$	0.0
x_2	0.437500	$g_2(x)$	-3.59E-02
x_3	42.098446	$g_3(x)$	0.0
x_4	176.636596	$g_4(x)$	-6.34E+01
f(x)	**6059.714**		
FE	24000		

3.4.4.5 Result of applying ISADE for constrained engineering optimization

Table.3.4, Table.3.5, Table.3.6 and Table.3.7 show the vectors of the best solution as well as the values of the constraint terms reached by ISADE, for each of the problems tested. We got the optimum solution at 8000 objective Function Evaluations (FE) per run with the welded beam problem, 24000 with the pressure vessel problem, 12000 with the speed reducer problem and 8000 with the tension/compression spring problem. We also tested the algorithm with more than above of objective function evaluation, but no performance improvements.

Table 3.6: Result of applying ISADE for E03(Speed reducer) problem.

Solutions		Constraints		Constraints	
x_1	3.500008	$g_1(x)$	-7.39E-02	$g_9(x)$	-5.83E-01
x_2	0.700000	$g_2(x)$	-1.98E-01	$g_{10}(x)$	-5.13E-02
x_3	17.000014	$g_3(x)$	-4.99E-01	$g_{11}(x)$	-1.09E-02
x_4	7.300033	$g_4(x)$	-9.01E-01	FE	15000
x_5	7.8000016	$g_5(x)$	-9.16E-06		
x_6	3.350225	$g_6(x)$	-3.08E-07		
x_7	5.286684	$g_7(x)$	-7.02E-01		
$f(x)$	**2996.358**	$g_8(x)$	-2.30E-06		

Table 3.7: Result of applying ISADE for E04 (Tension/Compression spring).

Solutions		Constraints	
x_1	0.051690	$g_1(x)$	0
x_2	0.356742	$g_2(x)$	0
x_3	11.287534	$g_3(x)$	-4.05E00
$f(x)$	**0.012665**	$g_4(x)$	-7.28E-01
FE	8000		

3.5 Conclusion

Locating global minimizers is a very challenging task for any minimization method. To overcome the weak point of EAs, and to achieve the global search for the solution space of multi-peak optimization problems with multi-dimensions, we proposed new evolution strategies of improvement of self-adaptive differential evolution is proposed. The main idea is that the three mutation scheme operators are chosen to be applied to individuals in the current population with the same probability, the scalar factor F is adaptively calculated by sigmoid function after ranking population in their fitness value and the control parameter CR is adjusted to balance the abilities of DE in exploitation and DE in exploration.

The search ability of ISADE with multi-dimensions optimization problems is very effective, compared with that of EAs. Nevertheless, the number of digits of design variables in the numerical experiments is insufficient to discuss about the stability of convergence.

Some benchmark test functions are used to validate the performance of the ISADE. The proposed approach performed well in several test problems both in terms of the number of fitness function evaluations required and in terms of the quality of the solutions found. The results show that ISADE outperforms in most function minimization. The experimental results showed that the accuracy and speed performance of this study had significantly outperformed the results produced by to other Evolutionary Algorithms (EAs), and Memetic Algorithms (MAs). Moreover, the convergence analysis showed that the proposed method was capable to escape from the local optima more effectively. We confirmed that these methods could reduce the calculation cost and dramatically improve the convergence towards the optimal solution. Moreover, it could solve large scale optimization problems with high probability.

This study plans to do a modified DE with the self-adaptive parameter in DE. To improve the local search ability of DE, we will present new hybrid ISADE with a local method name as Nelder_mead Simplex method in the next chapter and apply it for some real constrain engineering optimization.

Chapter 4

Training Artificial Feed-forward Neural Network using Modification of Differential Evolution Algorithm

In the chapter 3, we introduce the new version of DE algorithm, to show it's powerful this chapter we will apply this algorithm to training an artificial neural network (ANN). Training an artificial neural network (ANN) is an optimization task where the result is to find optimal weight and bias set of the network. There are many traditional method to training ANN, such as Back Propagation (BP) Algorithm, Levenberg-Marquadt(LM), Quasi-Newton(QN), Genetic Algorithm(GA) etc. Traditional training algorithms might get stuck in local minima and the global search techniques might catch global minima very slow. Therefore this research we apply the improvement of self-adaptive strategy for controlling parameters in differential evolution algorithm (ISADE) for training neural network.

4.1 Introduction

Artificial Neural Networks (ANNs) are widely applied in many fields of science, in pattern classification, function approximation, optimization, pattern matching and associative memories [33], [47]. Currently, there have been many algorithms used to train the ANNs, such as back propagation (BP) algorithm, Levenberg-Marquadt(LM), Quasi-Newton(QN), genetic algorithm (GA), simulating annealing (SA) algorithm, particle swarm optimization (PSO) algorithm, hybrid PSO-BP algorithm [36], hybrid ABC-BP algorithm [8] and so on. Back propagation (BP) learning can realize the training of feed-forward multilayer neural network. The algorithm mainly revises neural network weights according to the gradient descent methods to reduce error. This kind of method calculates simply. But there are still many drawbacks if neural network are used alone, for example, low training speed, easy to trap into local minimum point, and poor global searching ability, and so on. Though many improvements have already been carried on in this aspect, such as introducing momentum parameter, but it can't solve problem by the root. The ISADE [63] can overcome the barriers of BP algorithm.

In the last version of ISADE [63] we worked is to improve self-adaptive differential evolution, to do this the three DE's mutation scheme operators are selected as candidates due to their good performance on problems with different characteristics. These three mutation scheme operators are chosen to be applied to individuals in the current population with the same probability. The scaling factor F is calculated by ranking the population and applying formula of sigmoid function depend on the rank number of population size and the crossover control CR is also adaptively changed instead of taking fixed values to deal with different classes of problems. Another critical parameter of DE, the population size NP remains a user specified variable to tackle problems with different complexity.

The remainder of this chapter is organized as following manner. The concept of Artificial Neural Network is described in Section 4.1. Mathematical model of Neural Network and numerical experiments are described in Section 4.2. Finally, Section 4.3 includes some brief conclusions.

4.2 Training Feed-Forward Artificial Neural Network

4.2.1 Introduction Neural Network

NNs are mathematical modes that aim to represent certain characteristics of brain functions. This research was based on models of the living brain. Models of the brain are becoming increasingly significant owing to advances in neuroscience, especially the distinction between biology and neuroscience (also known as ANNs). NNs were first modelled by McCulloch and Pitts in 1943. The learning method was proposed by Hebb in 1949, which forms the basis of the current neural network learning method. The perceptron neural network was proposed in 1958. Recent studies have made further advances, including the development of computational resources, which have been reviewed by Yao [69], as follows.

- Incorporating the concept of energy into NNSs, which has a powerful effect when addressing combinatorial optimization problems.
- Exploiting the concept of physical annealing (simulated annealing), which is effective for solving various type of problems (Boltzmann model).
- A learning method for error back-propagation in the NN hierarchy [55] (back-propagation).

Neural network can be divided into three major learning paradigms such as supervised learning, unsupervised learning, and reinforcement learning. If supervised learning is that you are provided a mapping implied by the data, unsupervised learning is used for data clustering. As a result of any reduced dimension, for linearly inseparable problems and the amount of multi-dimensional data such as images and statistics, a good solution is relatively obtained with small amount of calculation. From this fact, including data mining and pattern recognition, have been applied in various fields. In reinforcement learning, data are ussually not given, but generated by an agent's interactions with environment. ANNs are frequently used in reinforcement learning as part of the overall algorithm.

4.2.1.1 Types of Neural Network

There are many types of Neural Networks (NN).

1. The feedforward neural network was the first and arguably most simple type of artificial neural network devised. In this network the information moves in only one direction forwards: from the input nodes data goes through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network. Feedforward networks can be constructed from different types of units. Continuous neurons, frequently with sigmoidal activation, are used in the context of backpropagation of error.

2. Contrary to feedforward networks, recurrent neural networks (RNNs) are models with bi-directional data flow. While a feedforward network propagates data linearly from input to output, RNNs also propagate data from later processing stages to earlier stages. RNNs can be used as general sequence processors.

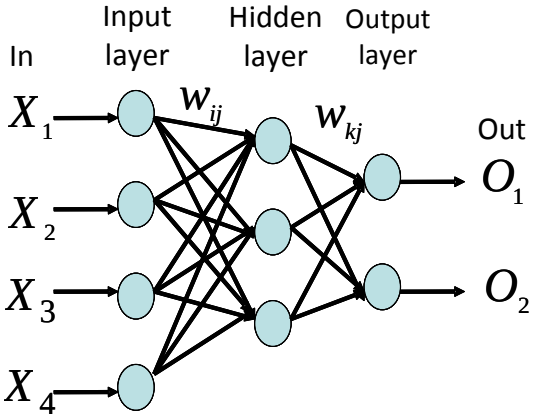

Figure 4.1: Hierarchical Neural Networks.

4.2.1.2 Neural Network Process

Back-propagation is a method used by the three-layer structure found in most NNs. Figure 4.1 shows the output of each neuron out_j, a sum of weights net_j,

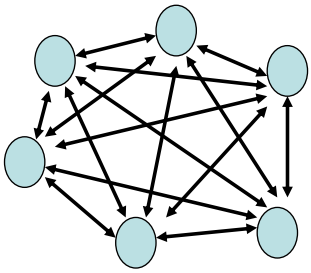

Figure 4.2: Neural Networks Interconnection.

the output unit connection weights from the hidden units w_{jk} and the connection weights to the hidden units from the input layer unit w_{ji}. The back-propagation error activation function (a sigmoid function) shown in equation (4.3) is often used. To determine whether the output of the output layer is much closer to the teacher signal, the error back-propagation method defines the squared error E, which can be expressed using equation (4.4). Thus, if E approaches 0, the output of the output layer approaches the teacher signal. Therefore, the purpose of the back-propagation method is to determine the weight of w_{ji}, w_{jk}.

$$net_j = \sum_{i=0}^{Ln} w_{ji} out_i \ , \tag{4.1}$$

$$w_{j^n i^n} = y_{nm} / y_{(n-1)m} \ , \tag{4.2}$$

$$f\left(net_j\right) = \frac{1}{1 + exp^{-net_j}} \ , \tag{4.3}$$

$$E = \frac{1}{2} \sum_{i}^{Ln} \left(y_{pi} - out_i\right)^2 ; (1 \leq p \leq P) \ . \tag{4.4}$$

4.2.1.3 Training Feed-Forward Artificial Neural Network

The neural network is a large-scale self-organization and self-adaptation nonlinear dynamic system. Artificial neural network technology is an effective way to solve

complex nonlinear mapping problem. In numerous neural network models, feed-forward multi-layer neural network model is one of the most widely used models in current, there are many researches show that three-layer feed-forward neural network can with arbitrary accuracy approximate any continuous function and its each order derivatives.

An ANN consists of a set of processing elements Fig. 4.3 also known as neurons or nodes, which are interconnected with each other [69]. In feed forward neural network models, shown in Fig. 4.4, each node receives a signal from the nodes in the previous layer and each of those signals is multiplied by a separate weight value. The weighted inputs are summed, and passed through a limiting function which scales the output to a fixed range of values. The output of the limiter is then broadcast to all of the nodes in the next layer. The input values to the inputs of the first layer, allow the signals to propagate through the network, and read the output values where output of the the node can be described by (4.5)

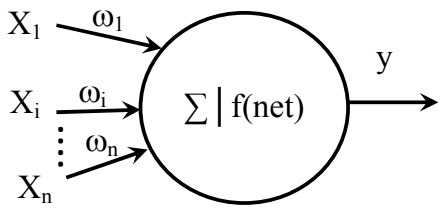

Figure 4.3: Processing unit of an ANN (neuron).

$$y_j = f_j(\sum_{i=1}^{n} w_{i,j}x_i + b_j) \tag{4.5}$$

where y_j is the output of node j, x_i is the ith input to the node j, w_{ij} is the connection weight between the node i and node j, b_j is the threshold (or bias) of the node j, and f_j is the node transfer function. Usually, the node transfer function is a nonlinear function such as a sigmoid function, a Gaussian function, etc. In this paper, the logarithmic sigmoid (4.6)

$$y = f(x) = \frac{1}{1 + e^{-x}} \tag{4.6}$$

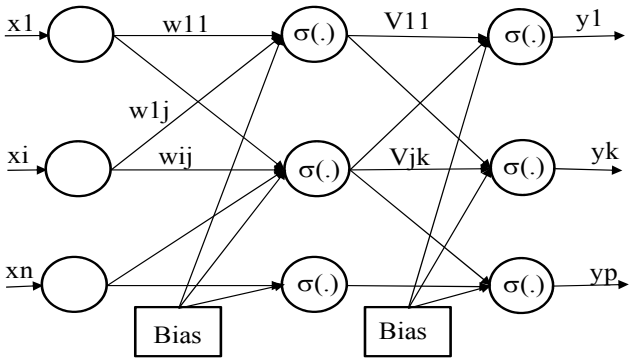

Figure 4.4: Multilayer feed-forward neural network (MLP).

The optimization goal is to minimize the objective function by optimizing the network weights. The mean square error (MSE), given by (4.7), is chosen as network error function.

$$E(\overrightarrow{w}(t)) = \frac{1}{N}\sum_{l=1}^{L}\sum_{k=1}^{p}(d_k - o_k)^2 \tag{4.7}$$

where $E(\overrightarrow{w}(t))$ is the error at the tth iteration; $\overrightarrow{w}(t)$ the weight vector at the tth iteration; d_k and o_k represent respectively the desired and actual values of kth output node; L is the number of patterns.

4.2.2 Numerical Experiments

We apply our ISADE to training some neural network, same in [8], that include XOR, 3-Bit Parity and Decoder-Encoder problems. These experiments involved 30 trials for each problem. The initial seed number was varied randomly during each trial.

The three layer feed-forward neural networks are used for each problem, i.e. one hidden layer and input and output layers. In the network structures, bias nodes are also applied and sigmoid function is placed as the activating function of the hidden nodes.

4.2.2.1 The Exclusive-OR Problem

The first test problem is the exclusive OR (XOR) Boolean function which is a difficult classification problem mapping two binary inputs to a single binary output shown in Table 4.1. In the simulations, we used a $2-2-1$ feed-forward neural network with six connection weights, no biases (having six parameters, XOR6) and a $2-2-1$ feed-forward neural network with six connection weights and three biases (having 9 parameters, XOR9) and a $2-3-1$ feed-forward neural network having nine connection weights and four biases totally thirteen parameters (XOR13). For XOR6, XOR9 and XOR13 problems, the parameter ranges $[-100, 100]$, $[-10, 10]$ and $[-10, 10]$ are used, respectively. The maximum iteration was 200.

Table 4.1: Binary XOR problem.

Input1	**Input2**	**Output**
0	0	0
0	1	1
1	0	1
1	1	0

4.2.2.2 The 3-Bit Parity Problem

The second test problem is the three bit parity problem. The problem is taking the modulus 2 of summation of three inputs. In other words, if the number of binary inputs is odd, the output is 1, otherwise it is 0 shown in Table 4.2. We use a $3-3-1$ feed-forward neural network structure for the 3-Bit Parity problem. The parameter range was $[-10, 10]$ for this problem. The maximum iteration was 400.

4.2.2.3 The 4-Bit Encoder-Decoder Problem

The third problem is 4-bit encoder/decoder problem. The network is presented with 4 distinct input patterns, each having only one bit turned on. The output

Table 4.2: 3-Bit parity problem.

Input1	**Input2**	**Input3**	**Out**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

is a duplication of the inputs shown in Table 4.3. A $4-2-4$ feed-forward neural network structure is used for this problem. For this problem, the parameter range is $[-50, 50]$. The maximum iteration was 400.

Table 4.3: 4-Bit Encoder-Decoder Problem.

Input1	**Input2**	**Input3**	**Input4**	**Out1**	**Out2**	**Out3**	**Out4**
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
1	0	0	0	1	0	0	0

4.2.3 Result of experiment

The results, given in Table 4.4, shows that the new algorithm ISADE chas faster convergence speed, and it can obtain the lesser mean square error, it is superior to the references. The convergence of the optimal solution could be improved more significantly in ISADE than that in references.

Table 4.4: Mean and standard deviation of MSE for algorithm and problems

Problem	Mean/std	ABC	ABC-LM	LM	ISADE
XOR6	Mean	0.007051	0.000752	0.110700	**1.1954E-21**
	std	0.00223	0.000980	0.063700	**5.3828E-21**
XOR9	Mean	0.006956	2.1246E-09	0.049100	**2.9189E-17**
	std	0.002402	1.9579E-10	0.064600	**1.4827E-16**
XOR13	Mean	0.006079	2.6111E-09	0.007800	**6.5278E-10**
	std	0.003182	1.2586E-09	0.022300	**3.5487E-09**
3-Bit Par	Mean	0.006679	6.3156E-07	0.020900	**5.3143E-15**
	std	0.002820	3.3189E-06	0.043000	**1.7173E-14**
Enc.Dec.	Mean	0.008191	1.3007E-06	0.024300	**9.8123E-17**
	std	0.001864	8.8443E-07	0.042400	**4.5439E-16**

4.3 CONCLUSIONS

Locating global minimizers is a very challenging task for any minimization method. In this research, a new improvement of self-adaptive differential evolution is proposed. The main idea is that the three mutation scheme operators are chosen to be applied to individuals in the current population with the same probability, the scalar factor F is adaptively calculated by sigmoid function after ranking population in their fitness value and the control parameter CR is adjusted to balance the abilities of DE in exploitation and DE in exploration.

The new algorithm ISADE is used to train feed-forward artificial neural networks on the XOR, 3- Bit Parity and 4-Bit Encoder-Decoder benchmark problems. The results of the experiments show that ISADE has better performance than the performance of the some reference algorithms.

Moreover, the ISADE was compared to other EAs, which showed that it was significantly better than other EAs.

We confirmed that the ISADE reduces the calculation cost and dramatically improves the convergence towards the optimal solution. It can also solve large scale optimization problems with a high probability.

The future work we will plan to apply this new algorithm ISADE for training neural networks on high dimensional classification and approximate benchmark problems.

Chapter 5

Hybrid Improved Self-Adaptive Differential Evolution and Nelder-Mead Simplex Method

In the chapter 3, we proposed a new improvement self-adaptive strategy for controlling parameters in differential evolution. The new algorithm shown it's outstanding in comparison with others evolutionary algorithms when running some benchmark test functions. However the new ISADE need to be improved the local search ability before applying to solve real structural optimization constraints. Therefore in this chapter we propose a new hybrid algorithm based on exploration power of a new improvement self-adaptive strategy for controlling parameters in differential evolution (ISADE) algorithm and exploitation capability of Nelder-Mead Simplex method is presented (HISADE-NMS) to reduce calculation cost, and to improve convergence towards the optimal solution. To valid the robustness of new hybrid algorithm, we apply it to solve some examples of structural optimization constraints. We confirmed satisfactory performance through various benchmark tests.

5.1 Introduction

Recently global optimization has been a concern of researchers, especially when the fitness function is dependent on a large number of variables or it is strictly confined to some constraints. A combination of two algorithms, in which one explores a promising area likely to contain global minima and the other exploits the area to find the desired point would be promising if properly performed. Global methods like particle swarm optimization, genetic algorithm, differential evolution, simulated annealing etc are known as efficient search engines to find and localize areas containing global minima, but they are very much time consuming while converging to a specified point. On the other hand local methods including Nelder-Mead Simplex [31], hill climbing, steepest descent, Newton Raphson method etc are good for exploitation of the search domain. In this study we present a hybrid algorithm based on exploitation (diversification) power of a new improvement of self-adaptive strategy for controlling parameters in differential evolution algorithm and exploitation (intensification) feature of the Nelder-Mead Simplex. The advantage of the simplex search method is that it is straightforward in an algorithmic sense and computationally efficient. However, as a result of using only local information, when they converge to a stationary point, there is no guarantee that the global optimum is found unless the domain in which the global minimum lies is provided. In contrast the DE [61] has been used in many practical cases and has demonstrated good convergence properties. DE explores the global search space without using local information of promising search directions. It has only a few control parameters as number of particles (NP), scaling factor (F) and crossover control (CR), which are kept fixed throughout the entire evolutionary process. However, these control parameters are very sensitive to the setting of the control parameters based on their experiments. The value of control parameters depend on the characteristics of each objective function, so we have to tune their value in each problem that mean it will take too long time to perform.

We propose a new hybrid algorithm based on exploration power of a new improvement self-adaptive strategy for controlling parameters in differential evo-

lution (ISADE) algorithm and exploitation capability of Nelder-Mead Simplex method is presented (HISADE-NMS) to reduce calculation cost, and to improve convergence towards the optimal solution.

5.2 What is a hybrid algorithm?

The best results found for various practical problems have proven that combination of different algorithms are very powerful, in case of large and difficult problems. Many hybrid metaheuristic algorithms have been proposed and implemented to solve many combinatorial optimization problems, e.g. those known as NP-hard. Ref [15] presented several hybridization methods for heuristic algorithms. According to Fig. 5.1, two algorithms can be hybridized at a high level or low level using relay or co-evolutionary methods, which may be homogeneous or heterogeneous.

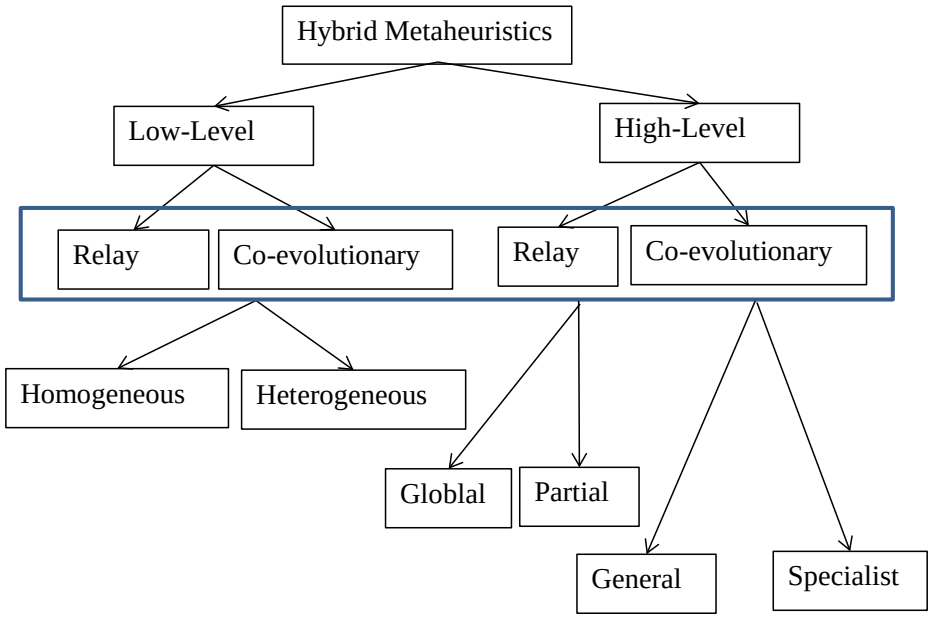

Figure 5.1: Classification of Hybrid Metaheuristic.

The low-level hybridization addresses the functional composition of a single optimization method. In this hybrid class, a given function of a metaheuristic is

replaced by another metaheuristic.

The high-level hybrid algorithms, the different metaheuristics are self-contained. We have no direct relationship to the internal workings of a metaheuristic.

Relay hybridization, a set of meta-heuristics is applied one after another, each using the output of the previous as its input, acting in a pipeline fashion.

Co-evolutionary hybridization represents cooperative optimization models, in which we have many parallel cooperating agents, each agent carries out a search in a solution space.

Four classes are divided from this hierarchical taxonomy:

LRH (Low-level Relay Hybrid). This class of hybrids represents algorithms in which a given metaheuristic is embedded into a single-solution metaheuristic. Few examples from the literature belong to this class.

LCH (Low-level Co-evolutionary Hybrid) Two competing goals govern the design of a metaheuristic: exploration and exploitation. In order to achieve the best performance, most efficient population-based heuristics (i.e., genetic algorithms, scatter search, ant colonies, etc.) have been coupled with local search method such as hill-climbing, simulated annealing and tabu search.

HRH (High-level Relay Hybrid). In HRH hybrid, self-contained metaheuristics are executed in a sequence. For example, evolutionary algorithms are not well suited for fine-tuning structures which are very close to optimal solutions. Instead, the strength of EA is in quickly locating the high performance regions of vast and complex search spaces. Once those regions are located, it may be useful to apply local search heuristics to the high performance structures evolved by the EA.

HCH (High-level Co-evolutionary Hybrid). The HCH scheme involves several self-contained algorithms performing a search in parallel, and cooperating to find an optimum. Intuitively, HCH will ultimately perform at least as well as one algorithm alone, more often perform better, each algorithm providing information to the others to help them.

5.3 Hybrid Improved Self-adaptive Differential Evolution and Nelder-Mead Simplex Method

follow chapter 3 to do more power of improve seft-adaptive differential evolution algorithm (ISADE) to reduce a large amount of calculation cost and to improve the convergence towards the optimal solution. We will present new hybrid ISADE with a local method name as Nelder_mead Simplex method in the next chapter and apply it for some real constrain engineering optimization.

5.3.1 Nelder-Mead Simplex Method

The NelderMead simplex search method (NMS) is based upon the work of J.A.Nelder and R.Mead [31]. A simplex is a geometrical figure consisting in n-dimensions, of (n+1) points: $x_1, \ldots, x_{n+1}$. Through a sequence of elementary geometric transformation (reflection, contraction, expansion and multi-contraction), the initial simplex moves, expands or contrasts. To select appropriate transformation, the method only uses the values of the objective function to be optimized at the verticals of the simplex considered. After each transformation, the current worst vertex is replaced by a better one. The trial movements in Fig. 5.2 to Fig. 5.2 are generated according to these operators (x_{n+1}: Represents the vertex where the objective function is the highest and x_l represents the vertex where the objective function is lowest).

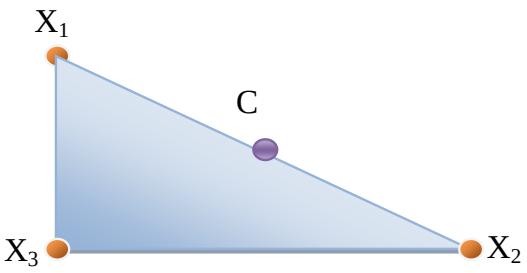

Figure 5.2: Simplex original in two dimensions.

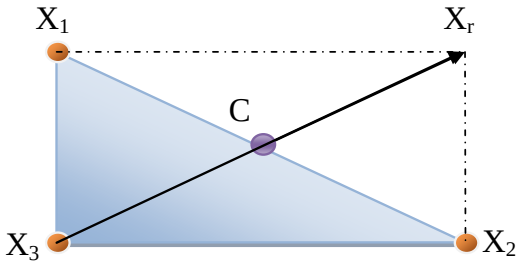

Figure 5.3: Simplex Reflection in two dimensions.

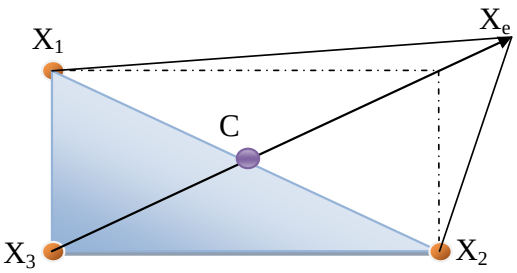

Figure 5.4: Simplex Expansion in two dimensions.

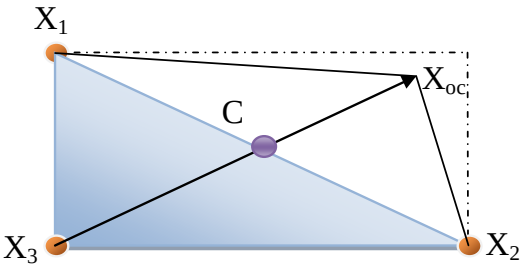

Figure 5.5: Simplex Outside contraction in two dimensions.

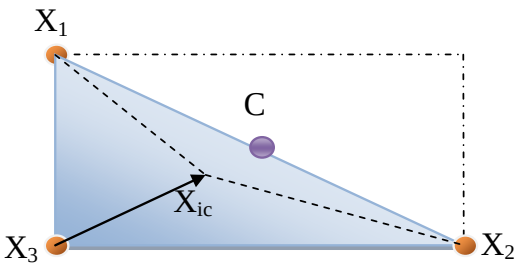

Figure 5.6: Simplex Inside contraction in two dimensions.

Algorithm 3 Nelder _ Mead algorithm

1: **Require:** α, β, γ and σ. Where: α, β, γ and σ are real parameters that control the operators of the Simplex.

2: INITIALIZE Creates initial population include $n+1$ points shown in Fig. 5.2

3: **while** (TERMINATION CONDITION) **do**

4: **Order:** Order the $n+1$ vertex to satisfy $f(X_1) \leq f(X_2) \leq \ldots \leq f(X_n) \leq f(X_{n+1})$ using the tie-breaking rules given below;

5: **Reflection:** Compute the reflection point X_r Fig. 5.3 shows reflection procedure

$$X_r = C + \alpha(C - X_{n+1}) = (1+\alpha)C - \alpha X_{n+1} \tag{5.1}$$

Where $C = \sum_{i=1}^{n} X_i/n$ is the centroid of the n best points (all vertices except for X_{n+1}). If $f_1 \leq f_r < f_n$, accept the reflected point X_r and terminate the iteration.;

6: **Expansion:** If $f_r < f_1$ then compute the expansion point X_e Fig. 5.4 shows expansion procedure.

$$X_e = C + \beta(X_r - C) = (1-\beta)C + \beta X_r = C + \alpha\beta(C - X_{n+1}) \tag{5.2}$$

If $f_e < f_r$, accept x_e and terminate the iteration; otherwise (if $f_e \geq f_r$), accept X_r and terminate the iteration. ;

7: **Contract:** if $f_r \geq f_n$ perform a contraction between and the better of X_{n+1} and X_r.

- Outside contraction: If $f_n \leq f_r < f_{n+1}$, perform an outside contraction: compute the outside contraction point Fig. 5.5 shows outside contraction procedure.

$$X_{oc} = C + \gamma(X_r - C) \tag{5.3}$$

If $f_{oc} \leq f_r$, accept X_{oc} and terminate the iteration; otherwise, go to step 8 (perform a shrink).

- Inside contraction: If $f_r \geq f_{n+1}$, perform an inside contraction: compute the inside contraction point X_{ic} Fig. 5.6 shows inside contraction procedure.

$$X_{ic} = C - \gamma(x_r - C) \tag{5.4}$$

8: **Perform a shrink step:** Fig. 5.7 shows inside contraction procedure. For $2 \leq i \leq n+1$ define:

$$X_i = X_1 + \sigma(X_i - X_1) \tag{5.5}$$

9: If the stopping conditions are not satisfied, repeat at step 4.

10: **end while**

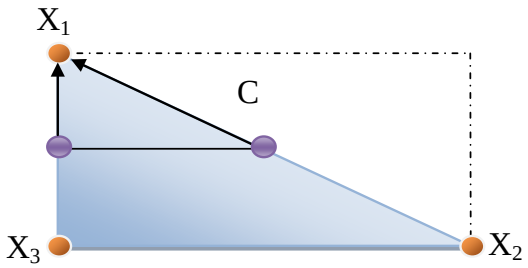

Figure 5.7: Simplex procedure shrink in two dimensions.

Fig. 5.2 to Fig. 5.7 show the effects of refection, expansion, contraction and shrinkage for a simplex in two dimensions (a triangle), using the standard coefficients $\alpha = 1$, $\beta = 2$, $\gamma = 0.5$ and $\sigma = 0.5$. Observe that, except in a shrink, the one new vertex always lies on the (extended) line joining of centroid C and x_{n+1}. Furthermore, it is visually evident that the simplex shape undergoes a noticeable change during an expansion or contraction with the standard coefficients.

5.3.2 Improve Self-adapting Control Parameters in Differential Evolution

In this section the hybrid algorithm HISADE-NMS is presented in detail.

5.3.2.1 Exploration of the Search Domain by Improving Self-adaptive Differential Evolution

The ISADE is ignited by the randomly selected particles. The algorithm takes these initial points to make up the first population in which the first generation will be born afterwards. Experiments show that a properly coded ISADE can explore the search domain to the global solution, although this magnitude is highly dependent on the complexity of the problem. We want to ignore the common convergence criteria of the ISADE by gradually decreasing the interference of the ISADE in the solution of the algorithm. According to this consideration, in the first steps, the ISADE is the only working algorithm that travels around the

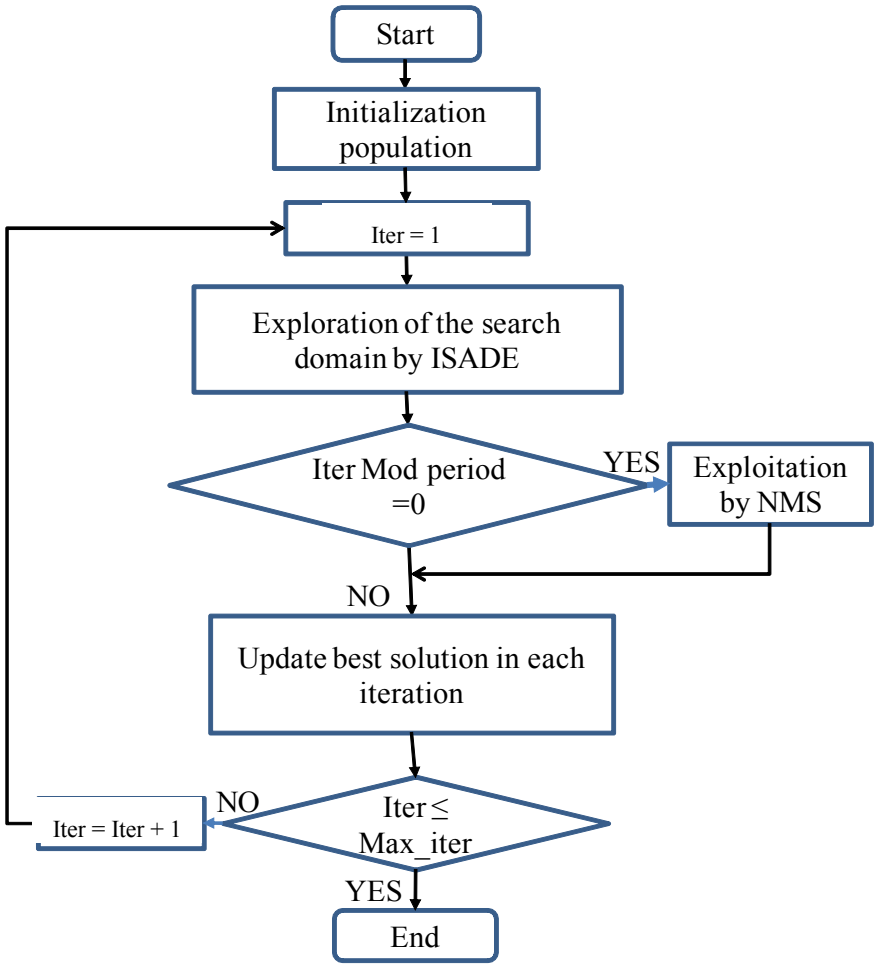

Figure 5.8: HISADE-NMS Procedure.

search domain to fine out the optimum regions and may be stopped if the maximum iteration is reached. Fig. 5.8 shows procedure of HIADE-NMS algorithm.

5.3.2.2 Exploitation Search Domain by Nelder-Mead Simplex Method

To improve the exploitation search ability of ISADE we will apply NMS after a number of iterations that is called period time. To do this we get D+1 best particle from main loop of ISADE as the D+1 vertex of NMS. After that the best particle, got from NMS, will be updated as best particle of main HISADE-NMS. The exploitation search of NMS can be seeing on Fig. 5.8.

5.4 Experiments

In this section, we will apply HISADE-NMS to solve some real constrained engineering design optimization problems to evaluate the performance of new hybrid algorithm HISADE-NMS. As in chapter 3, we used the following parameters: $NP = 8D$, $\tau = 0.1$. for ISADE and we use the standard coefficients $\alpha = 1$, $\beta = 2$, $\gamma = 0.5$ and $\sigma = 0.5$ for Nelder-Mead Simplex Method.

5.5 Result of applying HISADE-NMS for constrained engineering optimization

Table.5.1, Table.5.2, Table.5.3 and Table.5.4 show the vectors of the best solution as well as the values of the constraint terms reached by HISADE-NMS, for each of the problems tested. We got the optimum solution at 8000 objective Function Evaluations (FE) per run with the welded beam problem, 12000 with the pressure vessel problem, 12000 with the speed reducer problem and 8000 with the tension/compression spring problem. We also tested the algorithm with more than above of objective function evaluation, but no performance improvements.

Table 5.1: Result of applying HISADE-NMS for E01 (Welded beam) problem.

Solutions		Constraints	
x_1	0.205730	$g_1(x)$	-6.46E-05
x_2	3.470489	$g_2(x)$	-1.63E-04
x_3	9.036624	$g_3(x)$	-4.11E-09
x_4	0.205730	$g_4(x)$	-4.11E-09
f(x)	**1.724852**	$g_5(x)$	-4.11E-09
FE	8000	$g_6(x)$	-4.11E-09
		$g_7(x)$	-3.43E+00

Table 5.2: Result of applying HISADE-NMS for E02 (Pressure vessel) problem.

Solutions		Constraints	
x_1	0.812500	$g_1(x)$	0.0
x_2	0.437500	$g_2(x)$	-3.59E-02
x_3	42.098446	$g_3(x)$	0.0
x_4	176.636596	$g_4(x)$	-6.34E+01
f(x)	**6059.714**		
FE	12000		

From the result shown in Table.5.5, we can see that the new algorithm HISADE-NMS and original ISADE are both can get global optimization of all real constrained engineering design optimization problems, for the welded beam problem and the tension/compression spring problem both algorithms required same 8000 objective Function Evaluations (FE) to get final result. the new algorithm HISADE-NMS is better ISADE when running the pressure vessel problem and the speed reducer problem in comparison of objective function evaluations.

Table 5.3: Result of applying HISADE-NMS for E03(Speed reducer) problem.

Solutions		Constraints		Constraints	
x_1	3.500000	$g_1(x)$	-7.39E-02	$g_9(x)$	-5.83E-01
x_2	0.700000	$g_2(x)$	-1.98E-01	$g_{10}(x)$	-5.13E-02
x_3	17.000014	$g_3(x)$	-4.99E-01	$g_{11}(x)$	-1.09E-02
x_4	7.300035	$g_4(x)$	-9.01E-01	FE	12000
x_5	7.8000040	$g_5(x)$	-1.07E-01		
x_6	3.350215	$g_6(x)$	-9.68E-09		
x_7	5.286683	$g_7(x)$	-7.02E-01		
$f(x)$	**2996.349**	$g_8(x)$	-6.68E-08		

Table 5.4: Result of applying HISADE-NMS for E04 (Tension/Compression spring).

Solutions		Constraints	
x_1	0.051695	$g_1(x)$	-4.44E-16
x_2	0.356865	$g_2(x)$	-6.78E-14
x_3	11.280313	$g_3(x)$	-4.05E00
$f(x)$	**0.012665**	$g_4(x)$	-7.28E-01
FE	8000		

Table 5.5: Compare functional evaluation (FE) of HISADE-NMS and ISADE.

Problems	ISADE	HISADE-NMS
welded beam	8000	8000
pressure	24000	**12000**
speed reducer	15000	**12000**
tension\compression spring	8000	8000

5.6 Conclusion

Locating global minimizes is a very challenging task for any minimization method. In this research, a new hybrid improvement of self-adaptive differential evolution and Nelder-Mead simplex algorithm is proposed. We present a new version of the DE algorithm for obtaining self-adaptive control parameter settings. A hybrid algorithm using both of these characteristics (exploration and exploitation) would be promising to find global minima of problems. The main goals of the present hybrid algorithm are as follows:

-Reliability: A proper functioning of exploration and exploitation of the search domain (sometimes referred to as diversification and intensification) in order to find the true global minima.

-Efficiency: Using simple but effective combination to reduce the total amount of function evaluation. Some constrained engineering optimization are used to validate the performance of the HISADE-NMS. The proposed approach performed well in several test problems both in terms of the number of fitness function evaluations required and in terms of the quality of the solutions found.

Chapter 6

Conclusion

6.1 Contributions of This Dissertation

The overall objectives of these methodologies proposed in this dissertation are to solve large scale optimization problems, to reduce calculation cost, and to improve stability of convergence towards the optimal solution. Therefore, the approach that can lead to statistically significantly superior to other techniques is especially considered in this dissertation. The contributions of this dissertation are as follows:

Firstly, we propose the improvement self-adaptive for controlling parameters in differential evolution (ISADE) to solve large scale optimization problems, to reduce calculation cost, and to improve stability of convergence towards the optimal solution. These proposed algorithms combine the search ability of all optimization techniques, the global search ability and the local search ability of Adaptive Plan.

Secondly, new algorithms (ISADE) was applied to several numerical benchmark tests, constrained real parameter optimization and trained artificial neural network to evaluate its performance.

Finally, to improve the optimization process and overall performance of these methodologies, we introduce the hybridization of a local search algorithm with

an evolution algorithm (H-MNS_ISADE), which are the Nelder-Mead simplex method (MNS) and differential evolution (DE).

6.2 Future Work

In this dissertation, overcome the computational complexity, some integrated evolutionary strategy is proposed to solve large scale optimization problems, to reduce a large amount of calculation cost, and to improve the convergence to the optimal solution. Then, we verified the effectiveness of these algorithms by the numerical experiments performed some benchmark tests and constrained real parameter optimization.

Moreover, these methodologies was compared to other EAs, it shown to be statistically significantly superior to other EAs.

We confirmed that these algorithms reduces the calculation cost and dramatically improves the convergence towards the optimal solution. Moreover, it could solve large scale optimization problems with high probability.

About a solution of the problem of cost reduction, minimum time and maximum reliability, we would like to apply this study for optimal topology design shown in Fig. 6.1, it is our future work.

For the future research, several improvements are suggested to further enhance the performance of the proposed method. Firstly, the adaptive control parameter can be introduced to enhance the function evaluation scheme by the evolutionary operations. This is important to ensure that the speed performance will not be affected by the problem complexity. Lastly, the proposed method should be tested to estimate the parameters in more complex problems such as noise and identifiability.

Finally, we will apply this study to solve other constrained real parameters and dynamic optimization problems, and further real-life applications.

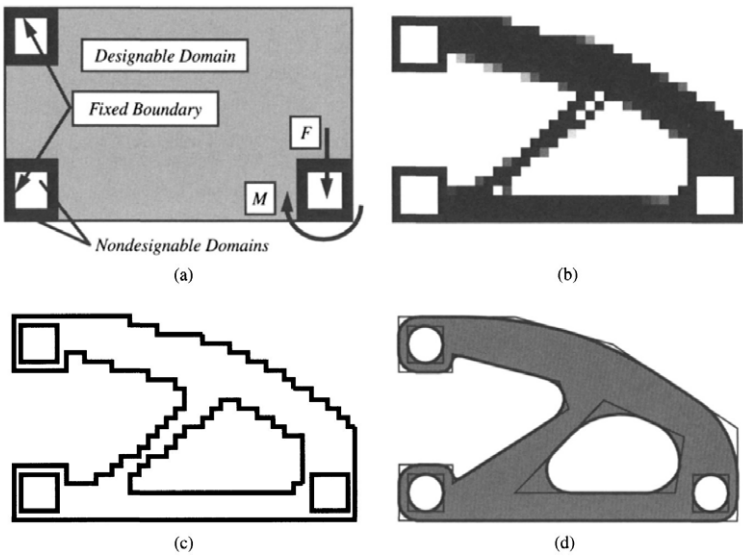

Figure 6.1: Optimal Topology Design.

Appendix

Optimization Benchmark Functions

To estimate the stability and convergence to the optimal solution, We will use 9 well-known benchmark test functions with 30 dimensions such as Sphere (Sp), Rosenbrock (Ro), Ridge (Ri), Griewank (Gr), Rastrigin (Ra), Ackley (Ac), Levy (Le), Schawefel (Sc) and Alpine (Al) function. These functions are given as follows

.1 Sphere Functions

Sphere functions is one of the simplest test benchmark. Function is continuous, convex and unimodal fig. 2 show the sphere function in 2D. It has the following general definition:

$$f_3 = \sum_{i=1}^{D} x_i^2 \tag{1}$$

where $x_i \in [-51.2, 51.2]$, $i = 1, \ldots, D$. Global minimum $f(x) = 0$ is obtainable for $x_i = 0$, $i = 1, \ldots, D$.

.2 Rosenbrock Functions

Rosenbrocks valley is a classic optimization problem, also known as banana function. The global optimum lays inside a long, narrow, parabolic shaped flat valley.

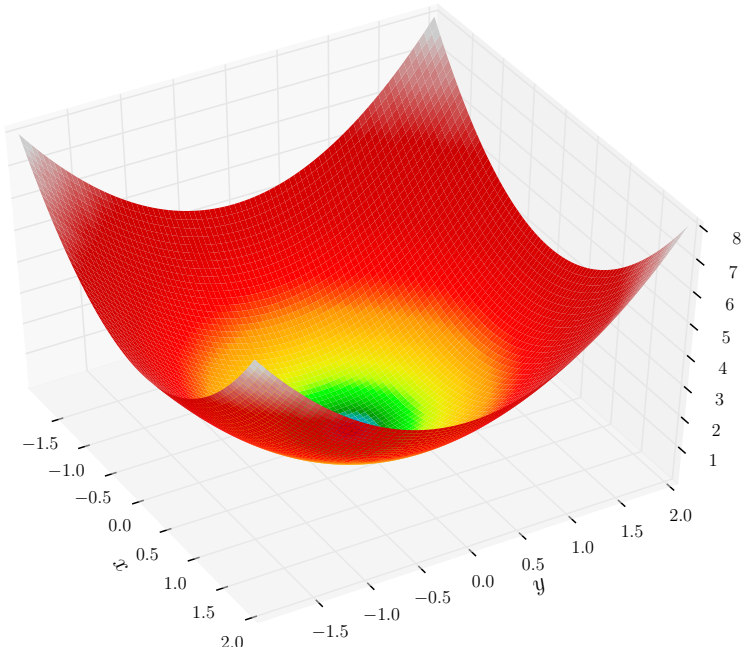

Figure 2: Sphere Functions in 2D.

To find the valley is trivial, however convergence to the global optimum is difficult and hence this problem has been frequently used to test the performance of optimization algorithms fig. 3 show the Rosenbrock function in 2D. Function has the following definition:

$$f_5 = \sum_{i=1}^{D} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \tag{2}$$

where $x_i \in [-2.048, 2.048]$, $i = 1, \ldots, D$. Global minimum $f(x) = 0$ is obtainable for $x_i = 1$, $i = 1, \ldots, D$.

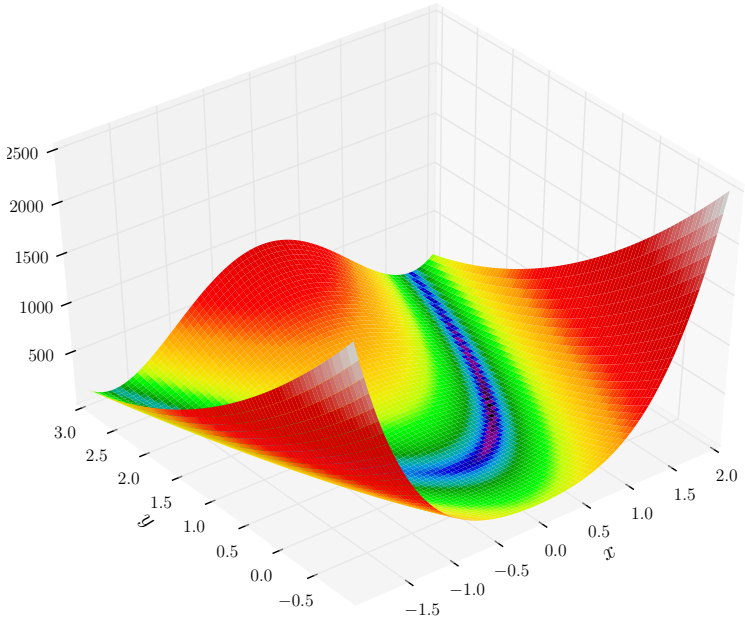

Figure 3: Rosenbrock Functions in 2D.

.3 Schwefels Problem 1.2 (Ridge Functions)

This function continuos, convex and unimodal, fig. 4 show the Schwefels Problem 1.2 in 2D. Function has the following definition:

$$f_3 = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{3}$$

where $x_i \in [-51.2, 51.2]$, $i = 1, \ldots, D$. Global minimum $f(x) = 0$ is obtainable for $x_i = 0$, $i = 1, \ldots, D$.

.4 Griewank Functions

Griewangks function is a non-convex function used as a performance test problem for optimization algorithms. The function interpretation changes with the scale; the general overview suggests convex function, medium-scale view suggests existence of local extreme, and finally zoom on the details indicates complex structure

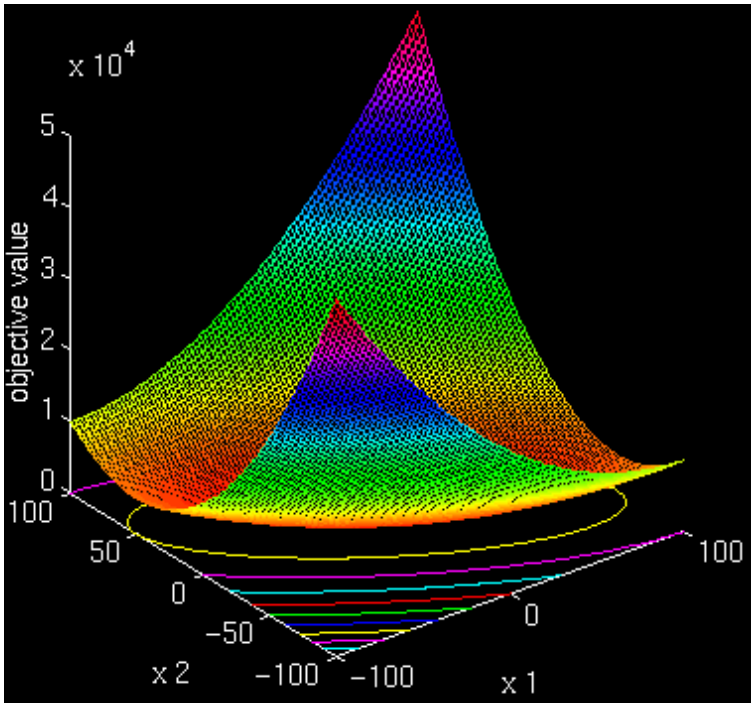

Figure 4: Ridge Functions in 2D.

of numerous local extreme, fig. 5 show the Griewank function in 2D. Function has the following definition:

$$f_{11} = 1 + \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{4}$$

where $x_i \in [-600, 600]$, $i = 1, \ldots, D$. Global minimum $f(x) = 0$ is obtainable for $x_i = 0$, $i = 1, \ldots, D$.

.5 Rastrigin Functions

The Rastrigin function is a non-convex function used as a performance test problem for optimization algorithms. It is a typical example of non-linear multimodal function. It was first proposed by Rastrigin [43] as a 2-dimensional function and has been generalized by Mühlenbein et al. [25] Finding the minimum of this

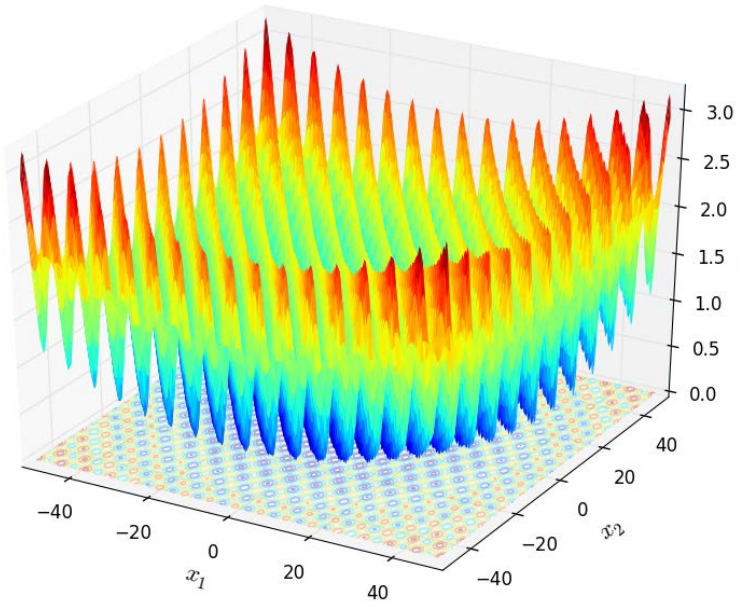

Figure 5: Griewank Functions in 2D.

function is a fairly difficult problem due to its large search space and its large number of local minima, fig. 6 show the Rastrigin function in 2D.

$$f_9 = 10D + \sum_{i=1}^{D}[x_i^2 - 10cos(2\pi x_i)] \tag{5}$$

where $x_i \in [-5.12, 5.12]$, $i = 1, \ldots, D$. Global minimum $f(x) = 0$ is obtainable for $x_i = 0$, $i = 1, \ldots, D$.

.6 Ackley Functions

The Ackley test function is multimodal and separable, with several local optima that, look more like noise, although they are located at regular intervals. The Ackley function only has one global optimum, fig. 7 show the Ackley function in

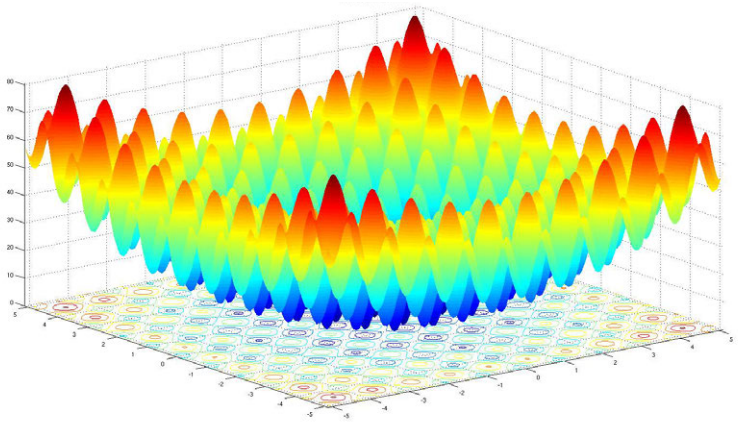

Figure 6: Rastrigin Functions in 2D.

2D.

$$f_{10} = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + e\ , \tag{6}$$

where $x_i \in [-51.2, 51.2]$, $i = 1, \ldots, D$. Global minimum $f(x) = 0$ is obtainable for $x_i = 0$, $i = 1, \ldots, D$.

.7 Levy Functions

Figure. 8 show the Levy function in 2D

$$f_7 = sin^2\left(3\pi x_1\right) + \sum_{i=1}^{n-1}\left(x_i - 1\right)\left(1 + sin^2\left(3\pi x_{i+1}\right)\right) + \left(x_n - 1\right)\left(1 + sin^2\left(2\pi x_n\right)\right) \tag{7}$$

where $x_i \in [-10, 10]$, $i = 1, \ldots, D$. Global minimum $f(x) = 0$ is obtainable for $x_i = 1$, $i = 1, \ldots, D$.

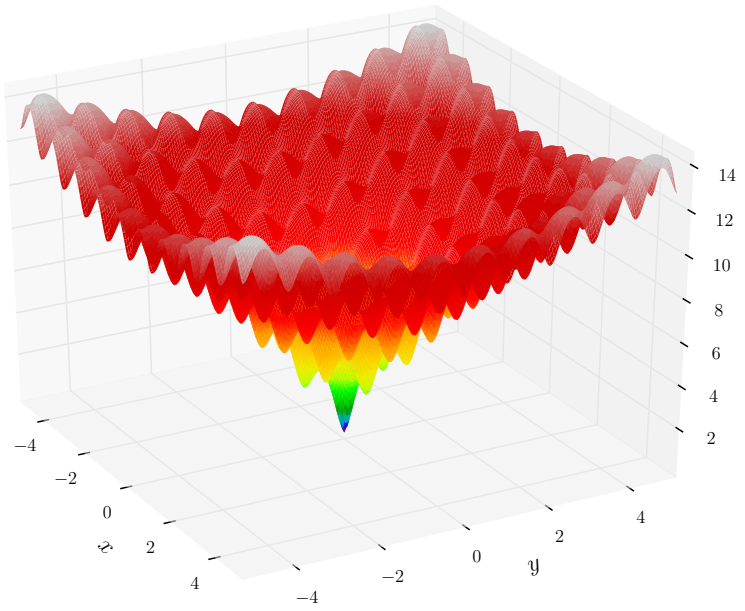

Figure 7: Ackley Functions in 2D.

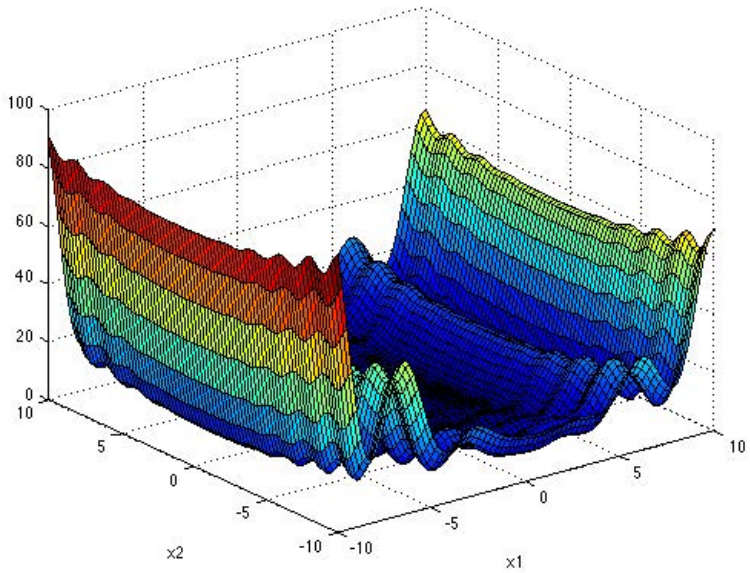

Figure 8: Levy Functions in 2D.

.8 Schawefel's problem 2.22

Figure. 9 show the Schawefel's problem 2.22 in 2D

$$f_8 = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i| \tag{8}$$

where $x_i \in [-10, 10]$, $i = 1, \ldots, D$. Global minimum $f(x) = 0$ is obtainable for $x_i = 0$, $i = 1, \ldots, D$.

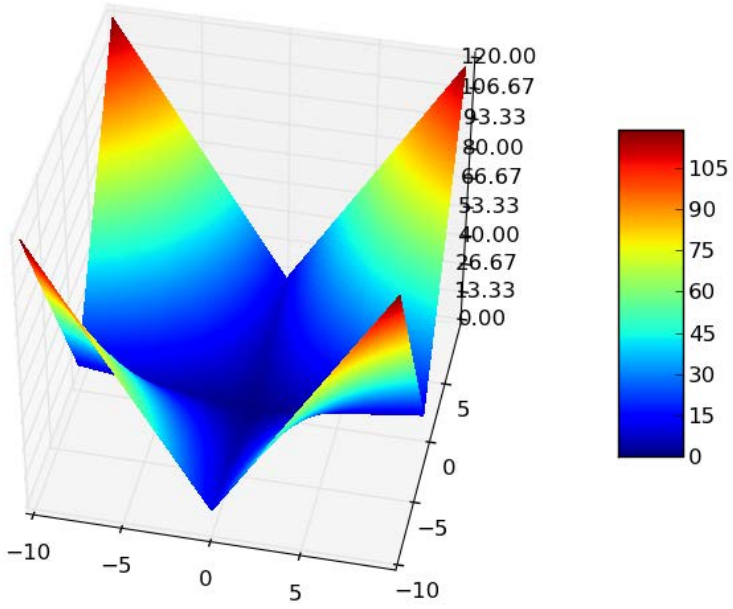

Figure 9: Schawefel's problem 2.22 in 2D.

.9 Alpine Functions

This is a multimodal minimization problem, fig. 10 show the Alpine function in 2D. The problem is defined as follows:

$$f_9 = \sum_{i=1}^{D} |x_i sin\,(x_i) + 0.1x_i| \tag{9}$$

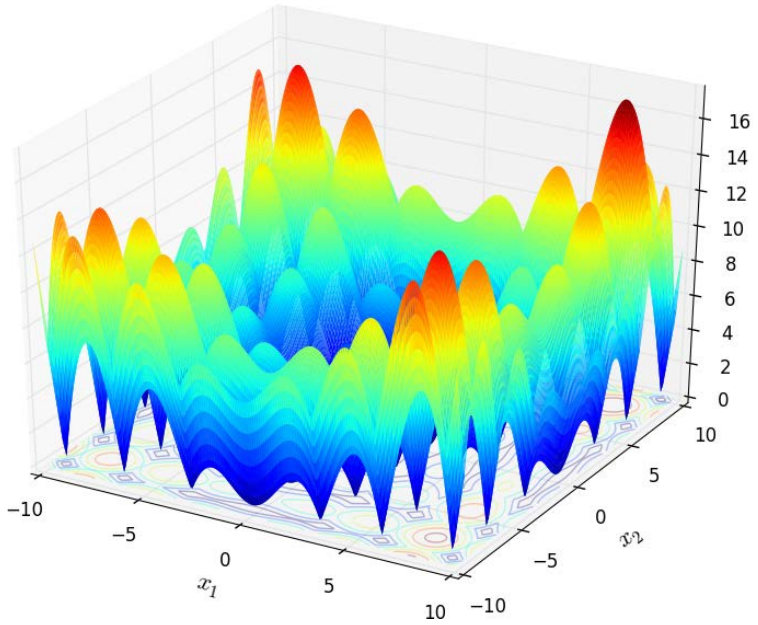

Figure 10: Alpine Functions in 2D.

where $x_i \in [-10, 10]$, $i = 1, \ldots, D$. Global minimum $f(x) = 0$ is obtainable for $x_i = 0$, $i = 1, \ldots, D$.

List of Publications

[P.1] T. Bui, H. Pham and H. Hasegawa, *"Improved Self-adaptive control parameters In Differential Evolution for solving constrained engineering optimization problems"*, Journal of Computational Science and Technology, Vol. 7, No. 1, pp. 59-74, 2013 April.

[P.2] T. Bui, H. Pham and H. Hasegawa, *"Hybrid Improved Self-adaptive Differential Evolution and Nelder-Mead Simplex method for solving constrained real-parameters"*, Journal of Mechanics Engineering and Automation. Vol.3 No.9 P.551-559, 2013 September.

[P.3] T. Bui and H. Hasegawa, *"Training Artificial Neural Network using Modification of Differential Evolution Algorithm"*, Journal of Machine Learning and Computing (IJMLC, ISSN: 2010-3700).Vol.5, No.1, pp.1-6, 2015 February.

[P.4] T. Bui, H. Pham and H. Hasegawa, *"Hybrid Integration of Differential Evolution with Articial Bee Colony for Global Optimization"*, 4th International Conference on Evolutionary Computation Theory and Applications (ECTA 2012), p. 15-23, 2012 October 5th -7th.

[P.5] T. Bui, H. Pham and H. Hasegawa, *"Modified Self-adaptive Strategy for Controlling Parameters in Differential Evolution"*, Asia Simulation Conference (AsiaSim), p. 370-378, 2012 October 27th-29th.

[P.6] T. Bui, H. Pham and H. Hasegawa, *"Hybrid Improved Self-adaptive Differential Evolution and Nelder-Mead Simplex method for solving constrained*

real-parameters", 5th International Conference on Manufacturing, Machine Design and Tribology (ICMDT), 2013.

[P.7] T. Bui and H. Hasegawa, *"Training Artificial Neural Network using Modification of Differential Evolution Algorithm*, 5th International Conference on Computer and Computational Intelligence (ICCCI), 2014.

References

[1] A.Baykasoglu and L.Ozbakr. Artificial bee colony algorithm and its application to generalized assignment problem, swarm intelligence: Focus on ant and particle swarm optimization. In *I-Tech Education and Publishing, Vienna, Austria*, 2007.

[2] A.Belegundu. A study of mathematical programming methods for structural optimization. In *PhD thesis*, pages –, Department of Civil Environmental Engineering, University of Iowa, Iowa, 1982.

[3] A.K.Qin and P.N.Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Evolutionary Computation, IEEE Congress (CEC2005)*, **2**, pages 1785–1791, 2005.

[4] Cascella G.L. Neri F. Salvatore N. Caponio, A. and M. Sumner. A fast adaptive memetic algorithm for online and offline control design of pmsm drives. *IEEE transactions on Systems, Man and Cybernetics Part B, Special Issue on Memetic Algorithms*, **37**(1):28–41, 2007.

[5] M. Clerc. *Particle swarm optimization.* ISTE, 2005.

[6] M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, **6**(1):58–73, 2002.

[7] M. Clerc and J. Kennedy. The particle swarm-explosion, stability and convergence in a multidimensional complex space. *IEEE Trans. Evol. Comput*, **6**(2):73–58, 2002.

[8] C.Ozturk and D.Karaboga. Hybrid artificial bee colony algorithm for neural network training. In *Evolutionary Computation (CEC)*. IEEE Congress, 2011.

[9] J.D. Digalakis and K.G. Margaritis. An experimental study of benchmarking functions for genetic algorithms. *Proceedings of IEEE Conference on Transactions*, **5**:3810–3815, 2000.

[10] D.Karaboga, B.Akay, and C.Ozturk. Artificial bee colony (abc) optimization algorithm for training feed-forward neural networks. In *Modeling Decisions for Artificial Intelligence*, 2007.

[11] D.Karaboga and B.Basturk. An artificial bee colony (abc) algorithm for numeric function optimization. In *IEEE Swarm Intelligence Symposium 2006, Indianapolis, Indiana, USA*, 2006.

[12] D.Karaboga and B.Basturk. A powerful and efficient algorithm for numerical function optimization:artificial bee colony (abc) algorithm. In *Journal of Global Optimization*, 2007.

[13] RC. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. **1**, pages 84–8, 2000.

[14] R.C. Eberhart and Y. Shi. Tracking and optimizing dynamic systems with particle swarms. pages 94–100, 2001.

[15] E.G.Talbi. A taxonomy of hybrid metaheuristic. In *Journal of Heuristics*, 2002.

[16] E.Sandgren. Nonlinear integer and discrete pro-gramming in mechanical design optimization. In *J.Mech. Des.-T. ASME*, **112**, pages 223–., 1990.

[17] A.V. Fiacco and G.P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley & Sons, New York., 1986.

[18] D.E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison - Wesley, 1989.

[19] Krasnogor N. Hart, W.E. and J.E. Smith. *Recent Advances in Memetic Algorithms*. Springer, 2005.

[20] H. Hasegawa. Adaptive plan system with genetic algorithm based on synthesis of local and global search method for multi-peak optimization problems. 2007.

[21] Sasaki H. Uehara H. Hasegawa, H. and K. Kawamo. The optimisation of spot-weld positions for vehicle design by using hybrid metaheuristics. *International Journal of Vehicle Design*, **43**(1-4):151–172, 2007.

[22] Yoshikawa M. Uehara H. Hasegawa, H. and K. Kawamo. The hybrid meta-heuristics by reflecting recognition of dependence relation among design variables for integer optimization of multi-peak problems. *Journal of Japan Society for Simulation Technology*, **25**(2):144–155, 2006.

[23] M.R Hestenes. Multiplier and gradient methods. In *Journal of Optimization Theory and Applications*, **4**, pages 303–320, 1969.

[24] Miki M. Hiroyasu, T. and M. Ogura. Parallel simulated annealing using genetic crossover. 2000.

[25] J.Born H.Mühlenbein and D.Schomisch. The parallel genetic algorithm as function optimizer. In *Parallel Computing*, number 17, pages 619–632, 1991.

[26] J. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. of Computing 2*, pages 88–105, 1973.

[27] J. Holland. Adaptation in natural and artificial systems. *The University of Michigan 1975, MIT Press*, 1992.

[28] J.H. Holland. Adaptation in natural and artificial systems. 1975.

[29] B. Bošković M. Mernik J. Brest, S. Greiner and V. ˇ Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Trans. Evol. Comput*, **10**(6):646–657, 2006.

[30] B. Bošković M. Mernik J. Brest, S. Greiner and V. ˇ Zumer. Performance comparison of self-adaptive and adaptive differential evolution algorithms. *Soft Comput*, **11**(7):617–629, 2007.

[31] J.A.Nelder and R.Mead. A simplex method for function minimization. In *Comput. J*, **7**, pages 308–313, 1965.

[32] J.Arora. Introduction to optimum design,. In *McGrawHill*, pages –, 1989.

[33] J.Dayhoff. Neural network architectures: An introduction. In *New York: Van Nostrand Reinhold*, 1990.

[34] J.Golinski. An adaptive optimization system applied to machine synthesis. In *Mech. Mach. Theory*, **8**, pages 419–436, 1973.

[35] J.Teo. Exploring dynamic self-adaptive populations in differential evolution. In *Soft Comput*, **10**, pages 673–686, 2006.

[36] J.Zhang, T.Lok, and M.Lyu. A hybrid particle swarm optimization back propagation algorithm for feed forward neural network training. In *Applied Mathematics and Computation*. ELSEVIER, 2007.

[37] Y.T. Kao and E. Zahara. A hybrid genetic algorithm and particle swarm optimization for multimodal functions. *Applied Soft Computing*, **8**(2):849–857, 2008.

[38] D. Karaboga. An idea based on honeybee swarm for numerical optimization. *In TECHNICAL REPORT-TR06*, 2005.

[39] J. Kennedy and R. Eberhart. Particle swarm optimization. **4**, pages 1942–1948, 1995.

[40] J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.

[41] Gelatt C. D. Vecchi M. P. Kirkpatrick, S. Optimization by simulated annealing. In *Science*, **220**, pages 671–680, Dec 1983.

[42] K.Ragsdell and D.Phillips. Optimal design of a class of welded structures using geometric programming. In *J. Eng. Ind*, **98**, pages 1021–1025, 1976.

[43] L.A.Rastrigin. Systems of extremal control. 1974.

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

[45] S.W. Mahfoud and D.E. Goldberg. A genetic algorithm for parallel simulated annealing. *Parallel Problem Solving from Nature*, (2):301–310, 1992.

[46] Hiroyasu T. Miki, M. and T. Fushimi. Parallel simulated annealing with adaptive neighborhood determined by ga. 2003.

[47] M.K, M.C, and R.S. Elements of artificial neural networks. In *Cambridge, MA: MIT Press*, 1997.

[48] Tirronen V. Kärkkäinen T. Neri, F. and T. Rossi. Fitness diversity based adaptation in multimeme algorithms: A comparative study. pages 2374–2381, 2007.

[49] Lim M.H. Zhu N. Ong, Y.S. and K.W. Wong. Classification of adaptive memetic algorithms: A comparative study. *IEEE transactions on Systems, Man and Cybnetics Part B*, **36**(1):141–152, 2006.

[50] Y.S. Ong and A.J. Keane. Meta-lamarckian learning in memetic algorithms. *IEEE transactions on evolutionary computation*, **8**(2):99–110, 2004.

[51] Panos Y. Papalambros and Douglass J. Wilde. *Principles of Optimal Design*. Modeling and Computation, The University of Michigan Press, Ann Arbor., 2nd edition edition, 2000.

[52] M.J.D. Powell. *A Method for Nonlinear Constraints in Minimization Problems, Optimization*. Edited by R. Fletcher, Academic Press, New York, New York., 1972.

[53] J.Blasco R.Meza, G.Sanchis and X.Herrero. Hybrid de algorithm with adaptive crossover operator for solving real-world numerical optimization problems. In *Evolutionary Computation (CEC2011), IEEE Congress*, pages 1551–1556, 2011.

[54] B.E. Rosen and R. Nakano. Simulated annealing - basics and recent topics on simulated annealing. *Journal of Japanese Society for Artificial Intelligence*, **9**(3), 1994.

[55] D.E. Rumelhart and J.L. McClelland. Parallel distributed processing: Explorations in the microstructure of cognition. *MIT Press*, 1986.

[56] Patrick Siarry Jayaraman V.K. Shelokar, P.S. and B.D. Kulkarni. Particle swarm and ant colony algorithms hybridized for improved continuous optimization. *Applied Mathematics and Computation*, **188**(1):129–142, 2007.

[57] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. pages 69–73, 1998.

[58] Y. Shi and RC. Eberhart. Empirical study of particle swarm optimization. **3**, pages 100–6, 1999.

[59] S.O.Soliman and T.L. Bui. A self-adaptive strategy for controlling parameters in differential evolution. In *IEEE World Congress on Computational Intelligence*, pages 2837–2842, 2008.

[60] D. Srinivasan and T.H. Seow. Evolutionary computation. pages 2292–2297, 2003.

[61] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. *In technical report tr-95-012. Technical report, ICSI*, 1995.

[62] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal Global Optimization*, **11**(4):341–357, 1997.

[63] Tam.BN, Hieu.PN, and H.Hasegawa. Improve self-adaptive control parameters in differential evolution for solving constrained engineering optimization problems. In *Journal of Computational Science and Technology*, 2013.

[64] Neri F. Kärkkäinen T. Majava K. Tirronen, V. and T. Rossi. An enhanced memetic differential evolution in filter design for defect detection in paper production. *Evolutionary Computation Journal*, **16**(4):529–555, 2008.

[65] S. Tooyama and H. Hasegawa. Adaptive plan system with genetic algorithm using the variable neighborhood range control. pages 846–853, 2009.

[66] H. Uehara. *Study on Development of General-purposed Optimization Engine and its Performance Evaluation - The proposal of Parallel Simulated Annealing with Selection.* Master's thesis, Shibaura Institute of Technology, 2004.

[67] Kawada H. Uehara, H. and K. Kawamo. Numerical experiments on optimal points searching using hybrid method of genetic algorithm and simulated annealing. pages 117–118, 2003.

[68] Amir Hossein Gandomi Xin-She Yang, Seyyed Soheil Sadat Hosseini. Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect. In *Comput. J*, **12**, pages 1180–1186, March 2012.

[69] X.Yao. Evolving artificial neural networks. **87**, pages 1423–1447, 1999.

[70] Xin-She Yang. *Nature-Inspired Metaheuristic Algorithms.* Luniver Press, University of Cambridge. United Kingdom., 2nd edition edition, 2010.

[71] Xin-She Yang. A new metaheuristic bat-inspired algorithm. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010) (Eds. J. R. Gonzalez et al.), Studies in Computational Intelligence, Springer Berlin,*, **284**, pages 65–74, Dec 2010.

[72] Xin-She Yang and S. Deb. Cuckoo search via lévy flights. In *Nature Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 210–214, Dec 2009.