Studies on Metaheuristics for Continuous Global Optimization Problems

Abdel-Rahman Hedar A. Ahmed

Studies on Metaheuristics for Continuous Global Optimization Problems

Abdel-Rahman Hedar A. Ahmed

Submitted in partial fulfillment of
the requirement for the degree of
Doctor of Informatics

Kyoto University
Kyoto, Japan
June 2004

Preface

The interface between computer science and operations research has drawn much attention recently especially in optimization which is a main tool in operations research. In optimization area, the interest on this interface has rapidly increased in the last few years in order to develop nonstandard algorithms that can deal with optimization problems which the standard optimization techniques often fail to deal with. Global optimization problems represent a main category of such problems. Global optimization refers to finding the extreme value of a given nonconvex function in a certain feasible region and such problems are classified in two classes; unconstrained and constrained problems. Solving global optimization problems has made great gain from the interest in the interface between computer science and operations research.

In general, the classical optimization techniques have difficulties in dealing with global optimization problems. One of the main reasons of their failure is that they can easily be entrapped in local minima. Moreover, these techniques cannot generate or even use the global information needed to find the global minimum for a function with multiple local minima. The interaction between computer science and optimization has yielded new practical solvers for global optimization problems, called *metaheuristics*. The structures of metaheuristics are mainly based on simulating nature and artificial intelligence tools. Metaheuristics mainly invoke exploration and exploitation search procedures in order to diversify the search all over the search space and intensify the search in some promising areas. Therefore, metaheuristics cannot easily be entrapped in local minima. However, metaheuristics are computationally costly due to their slow convergence. One of the main reasons for their slow convergence is that they may fail to detect promising search directions especially in the vicinity of local minima due to their random constructions.

In this study, both global optimization problem classes; unconstrained and constrained problems, are considered in the continuous search space. New hybrid versions of metaheuristics are proposed as promising solvers for the considered problems. The proposed methods

aim to overcome the drawbacks of slow convergence and random constructions of metaheuristics. In these hybrid methods, local search strategies are inlaid inside metaheuristics in order to guide them especially in the vicinity of local minima, and overcome their slow convergence especially in the final stage of the search.

Metaheuristics are derivative-free methods so that direct search methods, which are also derivative-free methods, are invoked to play the role of local search in the proposed hybrid methods. Therefore, the hybrid methods proposed in this study confront the growth of many optimization problems in which the gradient information is not available.

Kyoto, Japan — Abdel-Rahman Hedar A. Ahmed
June 2004

Acknowledgements

I am greatly indebted to my supervisor, Prof. Masao Fukushima, one of the great scientists in optimization and mathematical programming, for many things. I thank him for accepting me to be one of his students in Kyoto University from four years ago, for his suggestion of this research area to work on, and for his direct supervision. I would also like to thank him for his continual suggestions and support during this research. I owe special thanks for him for carefully reading and invaluable suggestions and corrections of the draft manuscripts of the parts of this thesis.

I would like to thank all members of Prof. Fukushima's research group, Prof. Tetsuya Takine, Dr. Nobuo Yamashita and my colleagues, for the good scientific atmosphere they offered to me during my study in Kyoto University. I would also like to thank Prof. Toshihide Ibaraki and Dr. Mutsunori Yagiura for providing me with some essential research papers needed in my study from their own libraries. Moreover, I am very grateful that I have been graduated from Kyoto University, one of the best universities in Japan and all over the world.

I owe thanks to Egyptian Government and Kyoto University for supporting my study in Japan. I am grateful for all research facilities that have been provided to me in Kyoto University to achieve this study.

Although I was very happy to be a great scientific environment represented in Kyoto University, it was hard to achieve the needed research atmosphere without my wife's support. Actually, it was not so easy for me to be settled in an oriental culture like Japanese Culture without her accompany. So, I am very grateful for her continual supports and encouragement. I owe great thanks to my great parents for all things that they gave to me or taught me. Without their supports I would never have made any success.

I have been brought up in an Islamic culture and environment in which my parents taught me importance of thanks and appreciation to others and primarily to Allah, the Creator, the Ultimate Source of all gifts in life. I have been always grateful to their teaching so lastly

and above all, I would thank Him Almighty for all His gifts, guidance and helps.

Contents

Chapter 1

Introduction

Many recent problems in science, engineering and economics can be expressed as computing globally optimal solutions [48, 49, 74, 75, 76]. Using classical nonlinear programming techniques may fail to solve such problems because these problems usually contain multiple local optima. Therefore, global search methods should be invoked in order to deal with such problems. In recent years, there has been a great deal of interest in emerging some artificial intelligence tools in the area of optimization. These tools which are normally called *metaheuristics* are mainly proposed by simulating nature or by invoking intelligent learned procedures [30, 73, 81].

One main category of global optimization problems contains the problems which are characterized by one or more of the following properties:

- Calculation of the objective function (or constraint functions if exist) is very expensive or time consuming.
- The exact gradient of the objective function (or constraint functions if exist) cannot be computed, or its numerical approximation is very expensive or time consuming.
- The values of the objective function (or constraint functions if exist) contain noise.

Such problems exist in many real-world applications and achieving the exact global solution is neither possible nor desirable. Therefore, using derivative-free global search methods is highly needed in order to achieve acceptable solutions. Actually, metaheuristics fight courageously when applied to these problems and they could obtain highly accurate solutions in many cases [73]. The power of metaheuristics comes from the fact that they are robust and can deal successfully with a wide range of problem areas. However, these methods, especially when they are applied to complex problems, suffer from the slow convergence that

brings about the high computational cost. The main reason for this slow convergence is that these methods explore the global search space by creating random movements without using much local information about promising search direction. In contrast, local search methods have faster convergence due to their using local information to determine the most promising search direction by creating logical movements. However, local search methods can easily be entrapped in local minima.

One approach that recently has drawn much attention is to combine metaheuristics with local search methods to design more efficient methods with relatively faster convergence than the pure metaheuristics, see [37, 38, 39, 40, 41, 42] and references therein. Moreover, these hybrid methods are not easily entrapped in local minima because they still maintain the merits of the metaheuristics.

In this study, new hybrid methods that combine metaheuristics and direct search methods are proposed in order to deal with the global optimization problems that have the above characteristics. Specifically, local search guidance in the direct search methods is invoked to direct and control the global search features of metaheuristics to design more efficient hybrid methods. In the rest of this chapter, some well-known direct search methods and metaheuristics are introduced to be used throughout this study. The mathematical definitions of the considered problems are given first.

1.1 Continuous Global Optimization Problems

In this study, both unconstrained and constrained global optimization problems in a continuous space are considered. Without loss of generality, only minimization problems are studied since maximization problems can be transformed to minimization problems by inverting the sign of their objective functions. The mathematical definitions for the considered problems are given below.

Unconstrained Problem

$$\min_{x \epsilon R^n} \; f(x), \tag{1.1.1}$$

where f is a generally nonconvex, real valued function defined on R^n.

Constrained Problem

$$\begin{aligned} \min_{x} \; & f(x), \\ \text{s.t.} \; & g_i(x) \leq 0, \; i = 1, \ldots, l, \\ & h_j(x) = 0, \; j = 1, \ldots, m, \\ & x \in \mathcal{S}, \end{aligned} \tag{1.1.2}$$

where f , g_i and h_j are real-valued functions defined on the search space $\mathcal{S} \subseteq R^n$. Usually, the search space $\mathcal{S}$ is defined as $\{x \in R^n : x_i \in [l_i, u_i]\,,\ i = 1, \ldots, n\}$.

1.2 Metaheuristics

The term "metaheuristics" was first proposed by Glover [27]. The word "metaheuristics" contains all heuristics methods that show evidence of achieving good quality solutions for the problem of interest within an acceptable time. Usually, metaheuristics offer no guarantee of obtaining the global solutions.

Metaheuristics can be classified into two classes; population-based methods and point-to-point methods. In the latter methods, the search invokes only one solution at the end of each iteration from which the search will start in the next iteration. On the other hand, the population-based methods invoke a set of many solutions at the end of each iteration. Below, we highlight the principles of genetic algorithm as an example of population-based methods, and simulated annealing and tabu search as examples of point-to-point methods.

1.2.1 Genetic Algorithms

A genetic algorithm (GA) is a procedure that tries to mimic the genetic evolution of a species. Specifically, GA simulates the biological processes that allow the consecutive generations in a population to adapt to their environment. The adaptation process is mainly applied through genetic inheritance from parents to children and through survival of the fittest. Therefore, GA is a population-based search methodology. Some pioneering works traced back to the middle of 1960s preceded the main presentation of the GAs of Holland [46] in 1975. However, GAs were limitedly applied until their multipurpose presentation of Goldberg [34] in search, optimization, design and machine learning areas. Nowadays, GAs are considered to be the most widely known and applicable type of metaheuristics [7, 8, 68].

GA starts with an initial population whose elements are called *chromosomes*. The chromosome consists of a fixed number of variables which are called *genes*. In order to evaluates and rank chromosomes in a population, a fitness function based on the objective function should be defined. Three operators must be specified to construct the complete structure of the GA procedure; selection, crossover and mutation operators. The selection operator cares with selecting an intermediate population from the current one in order to be used by the other operators; crossover and mutation. In this selection process, chromosomes with

higher fitness function values have a greater chance to be chosen than those with lower fitness function values. Pairs of parents in the intermediate population of the current generation are probabilistically chosen to be mated in order to reproduce the offspring. In order to increase the variability structure, the mutation operator is applied to alter one or more genes of a probabilistically chosen chromosome. Finally, another type of selection mechanism is applied to copy the survival members from the current generation to the next one.

GA operators; selection, crossover and mutation have been extensively studied. Many effective setting of these operators have been proposed to fit a wide variety of problems. More details about GA elements are discussed below before stating a standard GA in Algorithm 1.2.1.

Fitness Function

Fitness function F is a designed function that measures the goodness of a solution. It should be designed in the way that better solutions will have a higher fitness function value than worse solutions. The fitness function plays a major role in the selection process.

Coding

Coding in GA is the form in which chromosomes and genes are expressed. There are mainly two types of coding; binary and real. The binary coding was presented in the GA original presentation [46] in which the chromosome is expressed as a binary string. Therefore, the search space of the considered problem is mapped into a space of binary strings through a coder mapping. Then, after reproducing an offspring, a decoder mapping is applied to bring them back to their real form in order to compute their fitness function values. Actually, many researchers still believe that the binary coding is the ideal. However, the real coding is more applicable and easy in programming. Moreover, it seems that the real coding fits the continuous optimization problems better than the binary coding [44].

Selection

Consider a population P, selection operator selects a set $P' \subseteq P$ of the chromosomes that will be given the chance to be mated and mutated. The size of P' is the same as that of P but more fit chromosomes in P are chosen with higher probability to be included in P'. Therefore, the most fit chromosomes in P may be represented by more than one copy in P' and the least fit chromosomes in P may be not represented at all in P'.

Consider the population $P = \{x_1, x_2, \ldots, x_N\}$. The difference between selection operators lies in the way of computing the probability of including a copy of chromosome $x_i \in P$ into the set P', which is denoted by $p_s(x_i)$. Using these probabilities, the population is mapped onto a roulette wheel, where each chromosome x_i is represented by a space that proportionally corresponds to $p_s(x_i)$. Chromosomes in the set P' are chosen by repeatedly spinning the roulette wheel until all positions in P' are filled.

The most common selection operators are the *proportional selection* [46] and *linear ranking selection* [9], see [6] for ease of explanation and comparison of different selection operators. It is noteworthy that in the *proportional selection mechanism*, the probabilities $p_s(x_i),\ i = 1, \ldots, N$, are calculated by

$$p_s(x_i) = \frac{F(x_i)}{\sum_{j=1}^{N} F(x_j)}.$$

where F is the fitness function which must have positive values for all possible chromosomes in order to be used in this selection mechanism. In *linear ranking selection mechanism*, the chromosomes of P are sorted in the order of raw fitness, i.e.,

$$F(x_1) \leq F(x_2) \leq \cdots \leq F(x_N).$$

Then the probabilities $p_s(x_i),\ i = 1, \ldots, N$, are calculated by

$$p_s(x_i) = \frac{1}{M}\left(\eta_{\max} - (\eta_{\max} - \eta_{\min})\,\frac{i-1}{N-1}\right),$$

where $\eta_{\min} = 2 - \eta_{\max}$ and $1 \leq \eta_{\max} \leq 2$.

Crossover and Mutation

Crossover operator aims to interchange the information and genes between chromosomes. Therefore, crossover operator combines two or more parents to reproduce new children, then, one of these children may hopefully collect all good features that exist in his parents. Crossover operator is not typically applied for all parents but it is applied with probability p_c which is normally set equal to 0.6. Actually, crossover operator plays a major role in GA, so defining a proper crossover operator is highly needed in order to achieve a better performance of GA. Different types of crossover operators have been studied, see [44] as a condensed survey.

Mutation operator alters one or more gene in a chromosome. Mutation operator aims to achieve some stochastic variability of GA in order to get a quicker convergence. The probability p_m of applying the mutation operator is usually set to be small, normally 0.01.

Algorithm 1.2.1. *Genetic Algorithm*

1. ***Initialization.*** *Generate an initial population P_0. Set the crossover and mutation probabilities $p_c \in (0,1)$ and $p_m \in (0,1)$, respectively. Set the generation counter $t := 1$.*

2. ***Selection.*** *Evaluate the fitness function F at all chromosomes in P_t. Select an intermediate population P'_t from the current population P_t.*

3. ***Crossover.*** *Associate a random number from $(0,1)$ with each chromosome in P'_t and add this chromosome to the parents pool set S_t^P if the associated number is less than p_c. Repeat the following Steps 3.1 and 3.2 until all parents in S_t^P are mated:*

   3.1. *Choose two parents p_1 and p_2 from S_t^P. Mate p_1 and p_2 to reproduce children c_1 and c_2.*

   3.2. *Update the children pool set S_t^C through $S_t^C := S_t^C \cup \{c_1, c_2\}$ and update S_t^P through $S_t^P := S_t^P - \{p_1, p_2\}$.*

4. ***Mutation.*** *Associate a random number from $(0,1)$ with each gene in each chromosome in P'_t, mutate this gene if the associated number is less than p_m, and add the mutated chromosome only to the children pool set S_t^C.*

5. ***Stopping Conditions.*** *If stopping conditions are satisfied, then terminate. Otherwise, select the next generation P_{t+1} from $P_t \cup S_t^C$. Set S_t^C to be empty, set $t := t + 1$, and go to Step 2.*

1.2.2 Simulated Annealing

The original ideas of the simulated annealing (SA) methods dates back to 50s of the last century. Exactly, in 1953, Metropolis et al. [65] introduced an efficient algorithm to simulated the equilibrium of a collection of atoms at a given temperature. This pioneering technique

had inspired Kirkpatrik et al. [53] to simulate it in optimization and call it *Simulated Annealing (SA)*. Since the presentation of Kirkpatrik et al., a lot of studies that invoke SA have emerged in the area of optimization. Actually, the theoretical aspects as well as the applications of SA have been extensively studied, see [58, 57] and see [1, 43] as recent and short surveys.

The SA algorithm successively generates a trial point in a neighborhood of the current solution and determines whether or not the current solution is replaced by the trial point based on a probability depending on the difference between their function values. Convergence to an optimal solution can theoretically be guaranteed only after an infinite number of iterations controlled by the procedure so-called cooling schedule. The main control parameter in the cooling schedule is the temperature parameter T. The main role of T is to let the probability of accepting a new move be close to 1 in the earlier stage of the search and to let it be almost zero in the final stage of the search. A proper cooling schedule is needed in the finite-time implementation of SA to simulate the asymptotic convergence behavior of the SA. Algorithm 1.2.2 states the steps of the standard SA method.

Algorithm 1.2.2. *Simulated Annealing*

1. Initialization. *Choose an initial solution x_0, and fix the cooling schedule parameters; initial temperature $T_{\max}$, epoch length M, cooling reduction ratio $\lambda \in (0,1)$, and minimum temperature $T_{\min}$. Set the temperature $T := T_{\max}$ and $k := 0$.*

2. Epoch Loop. *Repeat the following steps (2.1–2.3) M times.*

2.1. *Generate a trial point y_k in the neighborhood of the current solution x_k.*

2.2. *Evaluate f on the trial point y_k, and compute*

$$p := \begin{cases} 1, & \text{if } f(y_k) < f(x_k); \\ \exp\left(\frac{-\Delta f}{T}\right), & \text{otherwise,} \end{cases}$$

where $\Delta f := f(y_k) - f(x_k)$.

2.3. *Choose a random number u from $(0,1)$. If $p \geq u$, set $x_{k+1} := y_k$. Otherwise, set $x_{k+1} := x_k$. Set $k := k + 1$.*

3. ***Termination Condition.*** *If the cooling schedule is completed ($T \leq T_{\min}$), terminate. Otherwise, decrease the temperature by setting $T := \lambda T$, and go to Step 2.*

One of the most powerful features of SA is its ability of easily escaping from being trapped in local minima by accepting up-hill moves through a probabilistic procedure especially in the earlier stages of the search. On the other hand, the main drawbacks that have been noticed on SA are its suffering from slow convergence and its wandering around the optimal solution if high accuracy is needed.

1.2.3 Tabu Search

Tabu Search (TS) is a heuristic method originally proposed by Glover in 1986 [27]. TS has been proposed and developed for combinatorial optimization problems [28, 29, 31]. TS fights courageously when applied to combinatorial optimization problems [31, 73, 81]. However, there is a very limit number of TS contributions in continuous optimization problems [39].

The main feature of TS is its use of an adaptive memory and responsive exploration. A simple TS combines a local search procedure with anti-cycling memory-based rules to prevent the search from getting trapped in local minima. Specifically, TS restricts returning to recently visited solutions by constructing a list of them called *Tabu List* (TL). In each iteration of the simple TS illustrated in Algorithm 1.2.3, TS generates many trial solutions in a neighborhood of the current solution. The trial solutions generation process is composed to avoid generating any trial solution that is already recently visited. The best trial solution found among the generated solutions will become the next solution. Therefore, TS can accept uphill movements to avoid getting trapped in local minima. TS can be terminated if the number of iterations without any improvement exceeds a predetermined maximum number.

Algorithm 1.2.3. *Simple Tabu Search*

1. *Choose an initial solution x_0. Set the Tabu List (TL) to be empty, and set the counter $k := 0$.*

2. *Generate neighborhood moves list $M(x_k) = \{x' : x' \in N(x_k)\}$, based on tabu restrictions, where $N(x_k)$ is a neighborhood of x_k.*

3. *Set x_{k+1} equal to the best trial solution in $M(x_k)$, and update TL.*

4. *If stopping conditions are satisfied, terminate. Otherwise, go to Step 2.*

A simple TS structure given in Algorithm 1.2.3 is called *short-term memory TS*. Updating the memory-based TL can be modified and controlled by the following concepts:

- **Tabu tenure:** number of iterations in which a tabu move is considered to remain tabu or forbidden;
- **Aspiration criteria:** accepting an improving solution even if generated by a tabu move.

The short-term memory is built to keep the recency only. In order to achieve better performance, long-term memory has been proposed to keep more important search features besides the recency, such as the quality and the frequency [32]. Specifically, long-term memory in high-level TS records attributes of special characters like elite and frequently visited solutions. Then, the search process of TS can adapt itself by using these special types of solutions in:

- **Intensification:** giving priority to elite solutions in order to obtain much better solutions in their vicinity.
- **Diversification:** discouraging attributes of frequently visited solutions in new move selection functions in order to diversify the search to other areas of solution space.

1.3 Direct search methods

Direct search methods can be simply defined as the procedures which try to direct the search for a minimum through the geometric intuition of the objective function by using function values only without evaluating the gradients, see [54, 78, 93]. In order to show the reality and the difficulty of the job that has been delegated to direct search methods, we borrow John Dennis' description of these methods which is stated and extended by Mike Powell in [77]:

> "It is to find the deepest point of a muddy lake, given a boat and a plump line, when there is a price to be paid for each sounding. A specification of an algorithm that is suitable for solving this problem would probably appeal to

> geometric intuition, and probably the procedure would require widely spaced measurements to be taken, in order to smooth out any high frequency variations in the depth of the lake. Experience has shown that many computer users find such algorithms attractive for a wide range of optimization calculations."

Direct search methods were originally proposed in the 1950s and 1960s to be justified in terms of geometric intuition in low dimensional spaces without mathematical proof. Since these methods are simple, easy to understand, easy to program, and widely applicable, they have remained popular for real-world problems in chemistry, chemical engineering and medicine. However, they failed to attract the mathematical optimization community until the appearance of their mathematical analysis from only fifteen years ago. In this section, Nelder-Mead and pattern search methods are presented as examples of direction search methods.

1.3.1 Nelder-Mead method

The local search method called the Nelder-Mead method [72] is one of the most popular derivative-free nonlinear optimization methods. Instead of using the derivative information on the function to be minimized, the Nelder-Mead method maintains at each iteration a nondegenerate simplex, a geometric figure in n dimensions of nonzero volume that is the convex hull of $n+1$ vertices, $x_1, x_2, \ldots, x_{n+1}$, and their respective function values. In each iteration, new points are computed, along with their function values, to form a new simplex. Four scalar parameters must be specified to define a complete Nelder-Mead method; coefficients of reflection ρ, expansion χ, contraction γ, and shrinkage σ. These parameters are chosen to satisfy

$$\rho > 0,\ \chi > 1,\ 0 < \gamma < 1,\ \text{and } 0 < \sigma < 1.$$

The Nelder-Mead method consists of the following steps:

Algorithm 1.3.1. *Nelder-Mead Method*

1. ***Order.*** *Order and re-label the $n+1$ vertices as $x_1, x_2, \ldots, x_{n+1}$ so that $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1})$. Since we want to minimize f, we refer to x_1 as the best vertex or point, to x_{n+1} as the worst point.*

2. Reflect. *Compute the reflection point* x_r *by*

$$x_r = \overline{x} + \rho\left(\overline{x} - x_{n+1}\right),$$

where $\overline{x}$ *is the centroid of the* n *best points, i.e.,* $\overline{x} = \sum_{i=1}^{n} x_i/n$. *Evaluate* $f(x_r)$. *If* $f(x_1) \leq f(x_r) < f(x_n)$, *replace* x_{n+1} *with the reflected point* x_r *and go to Step 6.*

3. Expand. *If* $f(x_r) < f(x_1)$, *compute the expansion point* x_e *by*

$$x_e = \overline{x} + \chi\left(x_r - \overline{x}\right).$$

Evaluate $f(x_e)$. *If* $f(x_e) < f(x_r)$, *replace* x_{n+1} *with* x_e *and go to Step 6; otherwise replace* x_{n+1} *with* x_r *and go to Step 6.*

4. Contract. *If* $f(x_r) \geq f(x_n)$, *perform a contraction between* $\overline{x}$ *and the better of* x_{n+1} *and* x_r.

4.1. Outside. *If* $f(x_n) \leq f(x_r) < f(x_{n+1})$ *(i.e.,* x_r *is strictly better than* x_{n+1}*), perform an outside contraction: Calculate*

$$x_{oc} = \overline{x} + \gamma\left(x_r - \overline{x}\right).$$

Evaluate $f(x_{oc})$. *If* $f(x_{oc}) \leq f(x_r)$, *replace* x_{n+1} *with* x_{oc} *and go to Step 6; otherwise, go to Step 5.*

4.2. Inside. *If* $f(x_r) \geq f(x_{n+1})$, *perform an inside contraction: Calculate*

$$x_{ic} = \overline{x} + \gamma\left(x_{n+1} - \overline{x}\right).$$

Evaluate $f(x_{ic})$. *If* $f(x_{ic}) \leq f(x_{n+1})$, *replace* x_{n+1} *with* x_{ic} *and go to Step 6; otherwise, go to Step 5.*

5. Shrink. *Evaluate* f *at the* n *new vertices*

$$x_i' = x_1 + \sigma\left(x_i - x_1\right), \quad i = 2, \ldots, n+1.$$

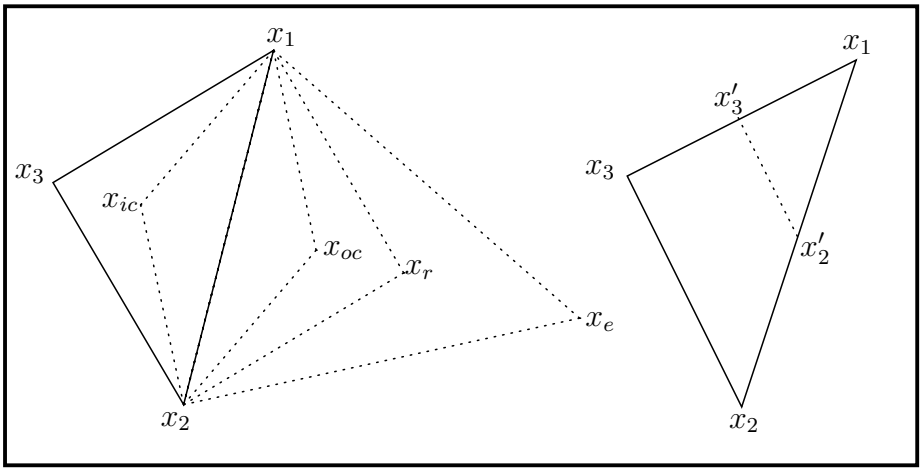

Figure 1.1: The reflection, expansion, contraction and shrinkage points for a simplex in two dimensions.

Replace the vertices $x_2, \ldots, x_{n+1}$ *with the new vertices* $x'_2, \ldots, x'_{n+1}$.

6. ***Stopping Condition.*** *Order and re-label the vertices of the new simplex as* $x_1, x_2, \ldots, x_{n+1}$ *such that* $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1})$. *If* $f(x_{n+1}) - f(x_1) < \varepsilon$, *then stop, where* $\varepsilon > 0$ *is a small predetermined tolerance. Otherwise, go to Step 2.*

Figure 1.1 shows the effects of reflection, expansion, contraction and shrinkage for a simplex in two dimensions using the standard values of the coefficients

$$\rho = 1,\ \chi = 2,\ \gamma = \frac{1}{2},\ \text{and}\ \sigma = \frac{1}{2}.$$

After more than thirty years of studying and applying the Nelder-Mead method, McKinnon [64] shows that the Nelder-Mead algorithm can stagnate and converge to a nonoptimal point even for very simple problems. However, Kelley [51, 52] proposes a test for sufficient decrease which, if passed for all iterations, will guarantee convergence of the Nelder-Mead iteration to a stationary point under some appropriate conditions. Moreover, he modified the Nelder-Mead method by invoking a reconstruction of the simplex called "*oriented restarts*" whenever the decrease test does not hold. This new orientation of the simplex is intended to

compensate for the kind of stagnation that was exhibited in [64]. However, this modification improves the robustness of the method, but it does not solve all the problems on stagnation of the Nelder-Mead method.

1.3.2 Pattern Search methods

Pattern search methods direct the search towards a local minimum through a pattern containing a certain number of points. Although the original pattern search algorithm has been proposed by Hooke and Jeeves [47], the pattern search methods as well as other direct search methods were not wildly applicable until fifteen years ago. The renaissance of pattern search methods began in 1989 with Torczon's Ph. D. thesis [87] and reached its mature stage by her paper [88] in which she has presented the *generalized pattern search* (GPS) as a framework of pattern search methods.

Pattern search methods invoke a pattern containing at least $n+1$ points in each iteration. One of these points is the current iterate and the other point are generated along search directions starting from the current iterate with a certain step size. The search directions used to generate the pattern points consist of a finite set of positive spanning directions in R^n. However, in order to achieve quicker convergence, other promising search directions may be included in addition to the positive spanning directions. Whenever the search fails to obtain a better movement, the step size is decreased in order to refine the search.

A sample GPS algorithm is stated in Algorithm 1.3.2 based on the one presented in [4]. In the initialization step of GPS, a set D of positive spanning directions in R^n should be chosen beside the initial solution and step size. For example, the set D can be set either $\{e_1, \ldots, e_n, -e_1, \ldots, -e_n\}$ or $\{e_1, \ldots, e_n, -e\}$, where $e_i \in R^n$ is the ith unit vector in R^n and $e \in R^n$ is the vector of ones. In each iteration of GPS, the mesh point set M_k and the poll points set P_k based on the set D should be computed as

$$M_k = \{y : y = x_k + \Delta_k dz \in X,\ d \in D,\ z \in Z_+^{|D|}\}, \tag{1.3.1}$$

$$P_k = \{y : y = x_k + \Delta_k d \in X,\ d \in D_k\}, \tag{1.3.2}$$

where Z_+ is the set of all positive integers. Moreover, the step size Δ_k is updated at each iteration in the way that it remains the same as its previous setting, or it is increased whenever an improvement is achieved, and it is decreased otherwise. More details about the step size updating process is given in [88] in order to fulfill some assumptions needed in the mathematical analysis of GPS.

The scenario of GPS starts with fitting the initial parameters, and then two search stages are invoked before the updating step. The first search stage is called *Search Step* in which any search procedure can be defined by the user to generate trial solutions from M_k. The main role of the *Search Step* is to achieve faster convergence of GPS. The other search stage called *Poll Step* is invoked as a systematic search in order to explore a region around the current solution. If an improvement is obtained, then the search is going on with the same step size or with a bigger step size if more promising solutions are expected. Otherwise, the current iterate is called a mesh optimizer and the step size is reduced in order to refine the mesh. GPS may be terminated when the step size becomes small enough.

Algorithm 1.3.2. *Generalized Pattern Search*

1. ***Initialization.*** *Choose an initial solution x_0, choose a positive spanning directions set D, choose a step size $\Delta_0 > 0$ and set the counter number $k := 0$.*
2. ***Search Step**: Compute the mesh M_k as in (1.3.1). Invoke a search strategy to get an improved point from M_k. If an improvement is obtained go to Step 4.*
3. ***Poll Step.*** *Choose the search direction set $D_k \subset D$ to be used in computing the poll set P_k as in (1.3.2). Evaluate f at all points in P_k.*
4. ***Update Step.*** *If an improved point obtained in Step 2 or 3, set x_{k+1} equal to this improved point, and set $\Delta_{k+1} \geq \Delta_k$. Otherwise, set $x_{k+1} := x_k$, and $\Delta_{k+1} < \Delta_k$.*
5. ***Termination Conditions.*** *If the termination conditions are satisfied, then stop. Otherwise, set $k := k + 1$, and go to Step 2.*

1.4 Organization and Contributions

In the subsequent chapters, we will introduce new hybrid methods that deal with the continuous global optimization problems in their two forms; unconstrained and constrained

problems. Below, we summarize the organization of the rest of the thesis as well as brief descriptions of the main contributions done in this study.

In Chapter 2, we give a new approach of hybrid direct search methods with metaheuristics of simulated annealing for finding a global minimum of a nonlinear function with continuous variables. First, we suggest a Simple Direct Search (SDS) method, which comes from some ideas of other well known direct search methods. Since our goal is to find global minima and the SDS method is still a local search method, we hybridize it with the standard simulated annealing to design a new method, called Simplex Simulated Annealing (SSA) method, which is expected to have some ability to look for a global minimum. To obtain faster convergence, we first accelerate the cooling schedule in SSA, and in the final stage, we apply Kelley's modification of the Nelder-Mead method on the best solutions found by the accelerated SSA method to improve the final results. We refer to this last method as the Direct Search Simulated Annealing (DSSA) method. The performance of SSA and DSSA is reported through extensive numerical experiments on some well known functions.

In Chapter 3, a new algorithm called Simplex Coding Genetic Algorithm (SCGA) is proposed by hybridizing genetic algorithm and Nelder-Mead method. In the SCGA, each chromosome in the population is a simplex and the gene is a vertex of this simplex. Selection, new multi-parents crossover and mutation procedures are used to improve the initial population. Moreover, Nelder-Mead method is applied to improve the population in the initial stage and every intermediate stage when new children are generated. Applying Nelder-Mead method again on the best point visited is the final stage in the SCGA to accelerate the search and to improve this best point. The efficiency of SCGA is tested on some well known functions.

In Chapter 4, we present a new approach of hybrid simulated annealing method for minimizing multimodel functions called the simulated annealing heuristic pattern search (SAHPS) method. Two subsidiary methods are proposed to achieve the final form of the global search method SAHPS. First, we introduce the approximate descent direction (ADD) method, which is a derivative-free procedure with high ability of producing a descent direction. Then, the ADD method is combined with a pattern search method with direction pruning to construct the heuristic pattern search (HPS) method. The last method is hybridized with simulated annealing to obtain the SAHPS method. The experimental results through well-known test functions are shown to demonstrate the efficiency of the SAHPS method.

In Chapter 5, we introduce a continuous TS called Directed Tabu Search (DTS) method. In the DTS method, direct-search-based strategies are used to direct a tabu search. These

strategies are based on the well-known Nelder-Mead method and a new pattern search procedure called adaptive pattern search. Moreover, we introduce a new tabu list conception with anti-cycling rules called Tabu Regions and Semi-Tabu Regions. In addition, Diversification and Intensification search schemes are employed. Numerical results of the DTS method are reported through extensive numerical experiments on several well known functions.

In Chapter 6, a simulated-annealing-based method called Filter Simulated Annealing (FSA) method is proposed to deal with the constrained global optimization problem. The considered problem is reformulated so as to take the form of optimizing two functions; the objective function and the constraint violation function. Then, the FSA method is applied to solve the reformulated problem. The FSA method invokes a multi-start diversification scheme in order to achieve an efficient exploration process. To deal with the considered problem, a filter-set-based procedure is built in the FSA structure. Finally, an intensification scheme is applied as a final stage of the proposed method in order to overcome the slow convergence of SA-based methods. The computational results obtained by the FSA method are reported and compared with some population-based methods.

Chapter 7 gives brief summary and conclusions of the main contributions in the thesis. Finally, the unconstrained test problems and the constrained test problems used throughout the study are given in Appendixes A and B, respectively.

Chapter 2

Direct Search SA for Unconstrained Global Optimization

2.1 Introduction

One approach that recently has drawn much attention is to combine simulated annealing (SA) method with local search methods to design more efficient methods with relatively faster convergence than the pure SA methods. Direct search methods, as local search methods, have got much attention in these combinations. For instance, SA was hybridized with simplex-based direct search method in [13, 79]. In addition, Kvasnicka and Pospichal [56] proposed a hybrid of controlled random search method, which is a generalization of the Nelder-Mead method, and SA.

In this chapter, we will hybridize SA and direct search methods to deal with the unconstrained optimization problem

$$\min_{x \epsilon R^n} \ f(x), \tag{2.1.1}$$

where f is a generally nonconvex, real valued function defined on R^n. First, we suggest a simple direct search (SDS) method, which comes from some ideas of other well known direct search methods. Since our goal is to find global minima and the SDS method is still a local search method, we hybridize it with the standard simulated annealing to design a new method, called simplex simulated annealing (SSA) method, which is expected to have some ability to look for global minima. The final method, called the direct search simulated annealing (DSSA) method, can be obtained by modifying SSA. To obtain faster convergence, we first accelerate the cooling schedule in SSA, and in the final stage, we apply Kelley's modification of the Nelder-Mead method [51, 52] on the best solutions found by the accelerated SSA to improve the final results. These two modifications on SSA will comprise

the final method DSSA. The performance of SSA and DSSA is reported through extensive numerical experiments on some well known functions. Comparing their performance with that of other metaheuristics methods shows that SSA and DSSA are promising in practice. Especially, DSSA is shown to be very efficient and robust.

To the author's knowledge, there are two main previous results on hybridizing simulated annealing with simplex methods. Press and Teukolsky [79] add a positive logarithmically distributed variable, proportional to the control annealing temperature T, to the function associated with every vertex of the simplex. Likewise, they subtract a similar random variable from the function value at every new replacement point. Then, their method may accept a new simplex whose actual function values at its vertices are not better than those at the previous simplex. This method was subsequently studied by Cardoso et al. [13, 14]. The other main result was presented by Kvasnicka and Pospichal [56]. Their method depends on the use of the simulated annealing acceptance in a controlled random search method. More precisely, the controlled random search uses a simplex method on randomly selected simplex sets from the population. So, to avoid being entrapped in local minima, they applied simulated annealing acceptance on the updating procedure. The common idea underlying these hybrid approaches and also our approach is to use simplex method to generate new logical movements while applying simulated annealing. However, the approach proposed in this chapter is different from the above mentioned approaches. We try to fix some disadvantages of simulated annealing like its slowness and its wandering near the global minimum in the final stage of search. So, we use a new simplex method to generate the movements trying to explore the function domain more carefully while applying accelerated simulated annealing, and also use another simplex method to accelerate the final stage in the search.

This chapter is organized as follows. In Section 2.2, we state the description of the proposed methods. Experimental results along with the initialization of some parameters and the setting of the control parameters of the proposed methods are discussed in Section 2.3. The conclusion of the contribution of this chapter makes up Section 2.4.

2.2 The description of the proposed methods

In this section, we describe the SDS, SSA and DSSA methods and introduce the initial and control parameters that are required by these methods. The values of these parameters used in the experiments will be given in Section 2.3.

2.2.1 Simple direct search (SDS)

Before we state the steps of the SDS method, we will introduce the main ideas which SDS comes from. The most famous simplex based direct search method was proposed by Nelder and Mead [72] in 1965. Nelder-Mead method has been studied extensively. In 1991, Dennis and Torczon [23] proposed a new form of direct search method, called the multidirectional search method, which can be considered an effective modification of Nelder-Mead method in the parallel computing environment. The main difference between Nelder-Mead method and the multidirectional search method is that the number of points used in the reflection step equals n in the latter method and equals one in Nelder-Mead method. Recently, Tseng [86] proposed a general framework of the simplex based direct search method which contains Nelder-Mead and the multidirectional search methods as subclasses and uses a varying number of reflected points in a flexible manner.

In the SDS method, we will start with an initial simplex with $n+1$ vertices. Then, we will try to get a better movement by reflecting the worst vertex in this simplex with respect to the remaining vertices. If the new vertex is not better than the worst one, we reflect the two worst vertices. If it fails to get a better point, then we reflect the three worst vertices and so on. If we reach the case of reflecting the n worst vertices and we still fail to get any better movement, then we will shrink the simplex.

Algorithm 2.2.1 below is a formal description of the SDS method. We require that the initial simplex S be a non-degenerate simplex with vertices $x_1, x_2, \ldots, x_{n+1}$. We assume throughout that the vertices are sorted according to the objective function values

$$f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1}). \tag{2.2.1}$$

We will refer to x_1 as the best vertex and x_{n+1} as the worst. Two scalar parameters ρ and σ that represent coefficients of reflection and shrinkage, respectively, must be specified to define the SDS method. We suppose that these parameters satisfy

$$\rho > 0,\ 0 < \sigma < 1. \tag{2.2.2}$$

Algorithm 2.2.1. *SDS(S, f, ϵ)*

1. ***Initialization.*** *Choose parameters ρ and σ satisfying (2.2.2). Select an initial simplex S with vertices $x_1, x_2, \ldots, x_{n+1}$. Choose a sufficiently small number $\epsilon > 0$.*

2. Order. *Order and re-label the vertices of S so that (2.2.1) holds.*

3. *If $f(x_{n+1}) - f(x_1) \leq \epsilon$, then terminate. Otherwise, go to Step 4.*

4. *Let $k := 1$. (k is the number of reflected points.)*

5. *If $k \leq n$, then go to Step 6 to perform the reflection. Otherwise, go to Step 7 to perform the shrinkage.*

6. Reflect. *Compute the k reflected points $\{x_i^r\}_{i=n-k+2}^{n+1}$ by*

$$x_i^r := \overline{x} + \rho(\overline{x} - x_i), \quad i = n+1, n, \ldots, n-k+2,$$

where $\overline{x}$ is the centroid of the set $\{x_1, x_2, \ldots, x_{n-k+1}\}$, i.e.,

$$\overline{x} := \frac{1}{n-k+1} \sum_{i=1}^{n-k+1} x_i. \tag{2.2.3}$$

Evaluate $f(x_i^r)$, $i = n+1, n, \ldots, n-k+2$. If $\min_{n-k+2 \leq i \leq n+1}\{f(x_i^r)\} < f(x_1)$, then put $x_i := x_i^r$, $i = n+1, n, \ldots, n-k+2$, and go to Step 2. Otherwise, let $k := k+1$ and go to Step 5.

7. Shrink. *Evaluate the function f at the n new vertices*

$$x_i := x_1 + \sigma(x_i - x_1), \quad i = 2, 3, \ldots, n+1. \tag{2.2.4}$$

Go to Step 2.

The coefficient of reflection ρ, in Algorithm 2.2.1, can be randomly chosen from the interval $(0.9, 1.1)$ to make more effective exploration. Algorithm 2.2.1 terminates when the function values at all the vertices become close to each other. However, if the number of iterations exceeds the predetermined allowed number of iterations, then we may terminate the algorithm.

SDS as well as other Simplex methods maintains at each iteration a nondegenerate simplex and the function values at the vertices. When one or more test points, along with their function values, are computed, we proceed to the next iteration with a new simplex. A most general approach of simplex methods was proposed by Tseng [86]. In this general approach,

an integer m $(1 \leq m \leq n)$ is chosen to specify the number of "good" vertices to be retained in constructing the initial trial simplices. The other vertices will be reflected, and then either expanded or contracted, at each iteration. If it fails to get a better point, then the whole simplex will be shrunk with respect to the best vertex. However, in Algorithm 2.2.1, we simply intensify the search by only repeating the reflection step in many directions and if it fails to get a better point, then we shrink the simplex with respect to the best vertex. It is noteworthy that the main aim of SDS is to enhance the exploration role to be a good seed to generate the global optimization methods SSA and DSSA. So, we will not compare the behavior of SDS with the other simplex based direct search methods in our experiments.

2.2.2 Simplex simulated annealing (SSA)

Since the SDS method is still a local search method, we hybridize it with the standard SA to perform simplex simulated annealing (SSA) method, which is expected to have the ability to look for a global minimum. We will apply this SA acceptance condition on the reflected points in the SDS method to obtain SSA method. In other words, we allow the possibility of accepting reflected points which do not include any better solution. Algorithm 2.2.2 describes the steps of SSA method and shows how we apply the simulated annealing acceptance and the cooling schedule with the lower limit temperature $T_{\min}$.

Algorithm 2.2.2. *SSA*$(S, f, \epsilon, T_{\min}, M)$

1. Initialization. *Choose parameter* $\sigma \in (0, 1)$. *Select an initial simplex* S *with vertices* $x_1, x_2, \ldots, x_{n+1}$. *Set the parameters of the cooling schedule: the initial temperature* T, $T_{\min}$ *and* M. *Choose a sufficiently small number* $\epsilon > 0$.

2. Order. *Order and re-label the vertices of* S *so that (2.2.1) holds.*

3. *If* $f(x_{n+1}) - f(x_1) \leq \epsilon$ *or* $T < T_{\min}$, *then terminate. Otherwise, go to Step 4.*

4. *Repeat the following Steps 4.1-4.5* M *times.*

4.1. *Let* $k := 1$.

4.2. *If* $k \leq n$, *then go to Step 4.3 to perform the reflection. Otherwise, go to Step 4.4 to perform the shrinkage.*

4.3. Reflect. *Compute the k reflected points $\{x_i^r\}_{i=n-k+2}^{n+1}$ by*

$$x_i^r := \overline{x} + \rho(\overline{x} - x_i), \quad i = n+1, n, \ldots, n-k+2,$$

where ρ is randomly chosen from the interval $(0.9, 1.1)$ and $\overline{x}$ is defined by (2.2.3). Evaluate $f(x_i^r)$, $i = n+1, n, \ldots, n-k+2$, and put $\widehat{f} := \min_{n-k+2 \le i \le n+1}\{f(x_i^r)\}$.

4.3.1. *If $\widehat{f} < f(x_1)$, then go to Step 4.3.3.*

4.3.2. *Compute $p := \exp\{-(\widehat{f} - f(x_1))/T\}$ and choose u randomly from the interval $(0, 1)$. If $p \ge u$, then go to Step 4.3.3. Otherwise, let $k := k + 1$ and go to Step 4.2.*

4.3.3. *Set $x_i := x_i^r$, $i = n+1, n, \ldots, n-k+2$. Go to Step 4.5.*

4.4. Shrink. *Shrink the simplex by determining n vertices by (2.2.4). Go to Step 4.5*

4.5. Sort. *Sort the vertices of S so that (2.2.1) holds.*

5. *Reduce the temperature T and go to Step 3.*

In Algorithm 2.2.2, the coefficient of reflection ρ is determined by choosing a random number from the interval $(0.9, 1.1)$. SSA method terminates when the function values at the vertices are close to each other or the cooling schedule is completed. The main role of M, the number of inner iterations per each temperature, is to get closer to the equilibrium because it has been proved [57, 58] that when M is sufficiently large and the temperature T is slowly reduced, the solution x will eventually be frozen at the global minimum.

2.2.3 Direct search simulated annealing (DSSA)

It is known that the standard SA may quickly approach the neighborhood of the global minimum but has a difficulty in obtaining some required accuracy. So, it is suitable to finish the algorithm with a faster convergent method. According to this idea, we modify SSA method to obtain the DSSA method as follows:

1. Accelerate the cooling schedule in SSA, i.e., use a smaller reduction factor for the temperature T.

2. Set the coefficient of shrinkage σ equals one to maintain the size of the initial simplex large enough. Actually, setting $0 < \sigma < 1$ is effective for achieving good behavior near a minimum in SSA, especially, in the final stage of search. However, in DSSA, the situation is different because we use the simplex simulated annealing part in exploring the whole domain and storing the best visited point in a list. So, perfect behavior near a minimum is not pursued in this part but it will be considered in the last part of DSSA using a complete local search method starting from each point in the best point list. In fact, it is known that local search methods have much better behavior near a minimum than global methods.

3. Store the best solutions found by the accelerated SSA in a list called "best list" as mentioned earlier and apply another local search method starting from each element of the best list to improve further these best solutions.

According to these modifications of SSA, we can state the steps of the DSSA method in Algorithm 2.2.3.

Algorithm 2.2.3. *DSSA*$(S, f, \epsilon, T_{\min}, M)$

1. Initialization. *Select an initial simplex S with vertices $x_1, x_2, \ldots, x_{n+1}$. Set the parameters of the cooling schedule: the initial temperature T, $T_{\min}$ and M. Set the size of the best list. Choose a sufficiently small number $\epsilon > 0$.*

2. Order. *Order and re-label the vertices of S so that (2.2.1) holds.*

3. Best list. *Store the m best points in the best list.*

4. *If $f(x_{n+1}) - f(x_1) \leq \epsilon$ or $T < T_{\min}$, then go to Step 7. Otherwise, go to Step 5.*

5. *Repeat the following Steps 5.1-5.4 M times.*

5.1. *Let $k := 1$.*

5.2. Reflect. *Compute the k reflected points $\{x_i^r\}_{i=n-k+2}^{n+1}$ by*

$$x_i^r := \overline{x} + \rho(\overline{x} - x_i), \quad i = n+1, n, \ldots, n-k+2,$$

where ρ is randomly chosen from the interval $(0.9, 1.1)$ and $\overline{x}$ is defined by (2.2.3). Evaluate $f(x_i^r)$, $i = n+1, n, \ldots, n-k+2$, and put $\widehat{f} := \min_{n-k+2 \le i \le n+1}\{f(x_i^r)\}$.

5.2.1. *If $\widehat{f} < f(x_1)$, then go to Step 5.2.3.*

5.2.2. *Compute $p := \exp\{-(\widehat{f} - f(x_1))/T\}$ and choose u randomly from the interval $(0, 1)$. If $p \ge u$, then go to Step 5.2.3. Otherwise, let $k := k + 1$ and go to Step 5.4.*

5.2.3. *Set $x_i := x_i^r$, $i = n+1, n, \ldots, n-k+2$. Go to Step 5.3.*

5.3. Sort. *Sort the vertices of S so that (2.2.1) holds and update the best list.*

5.4. *If $k \le n$, then go to Step 5.2.*

6. *Reduce the temperature T and go to Step 3.*

7. *From each point in the best list, construct a smaller simplex. Then, apply Kelley's modification of the Nelder-Mead method on each of these simplices.*

In the DSSA method, we use the Kelley's modification [51] of the Nelder-Mead method to refine the points stored in the best list.

2.3 Experimental results

The performance of SDS, SSA and DSSA methods has been evaluated to show how simulated annealing can affect the local search method SDS toward its generalization in global optimization. Moreover, the comparison between the results of SSA and DSSA shows the effect of the acceleration of convergence to improve the final results. Finally, the performance of our final method DSSA has been compared with some other metaheuristics methods. The

comparison was made using a set of some well known functions, which are listed in Appendix A.

2.3.1 Setting of parameters

Some initial parameters and control parameters must be specified to define the complete implementation of the methods SDS, SSA and DSSA.

Choosing the initial simplex

First, we randomly choose an initial orientation x_1 from some predetermined range of initial points for each function. Then, we take a step in each coordinate direction, called the edge of the simplex, to construct a right-angled simplex with vertices $x_1, x_2, \ldots, x_{n+1}$. The edge length of the simplex is chosen to fit the range of initial points for each function. For all test functions the edge length is varied from 0.125 to 4 depending on the range of initial points of each function. Moreover, we start with some suitable edge length and if the difference of the functional values at the simplex vertices is very small, then this edge length will be doubled until we get an improvement on the condition of the initial simplex or reach the maximum allowed edge length. This method of choosing the initial simplex is applied on all methods SDS, SSA and DSSA.

The cooling schedule

The cooling schedule consists of the initial temperature $T_{\max}$, the cooling function, the epoch length M and the stopping condition. As in Kirkparick et al. [53], we choose the value of $T_{\max}$ large enough to make the initial probability of accepting transition close to 1. We set the initial probability equal to 0.9. Then, $T_{\max}$ is calculated from the equation

$$T_{\max} = -\frac{f(x_{n+1}) - f(x_1)}{\ln(0.9)}.$$

The temperature is reduced with a so-called cooling function F, i.e., the temperature at the kth epoch is determined with $T_k = F(T_{k-1})$. In the standard SA, this equation will be $T_k = \alpha T_{k-1}$, where $\alpha \in [0.5, 0.99]$ is a parameter called cooling ratio. In SSA algorithm, we set $\alpha = 0.9$. Since the DSSA algorithm is designed by accelerating SSA, we set $\alpha = 0.5$, and the computational experience shows that this value of α gives good results for most of the test functions. However, for some hard functions; Shekel functions, Shubert function

and Griewank function, we have observed that it is more effective to slow down the cooling schedule by setting $\alpha = 0.7$. Epoch length M is the number of trials allowed at each temperature and we set it equal to $10n$ in SSA and n in DSSA. Finally, the stopping condition is comprised of the minimum allowed temperature $T_{\min}$ which equals $10^{-5} \times T_{\max}$ in both methods SSA and DSSA.

Termination criteria

The termination criteria of SDS, SSA and DSSA algorithms are intended to reflect the progress of these algorithms. So, we terminate these algorithms when the function values at all the vertices become close to each other, i.e.,

$$f(x_{n+1}) - f(x_1) \leq \epsilon,$$

where the tolerance ϵ is a small positive number and we set it 10^{-6} in SDS, SSA and 10^{-8} in DSSA. Moreover, SSA algorithm and the simulated annealing part of the DSSA algorithm can also be terminated if the cooling schedule is completed. However, if the number of iterations exceeds the predetermined allowed number of iterations, then we may terminate the algorithms. This maximum number equals $50n$ in SDS and DSSA and equals $1000n$ in SSA. We remark that, for Easom function, DSSA has had some difficulty in finding its minimum because it lies in a very narrow hole and outside this narrow hole the graph the function is almost flat. Termination before reaching this narrow hole could be avoided by repeating the algorithm with reducing the edge length of the simplex in each time until we get a very small edge length equal to 10^{-4}.

Best list

The remaining parameter is the number of the best points stored during the search in the DSSA method. This parameter is set equal to n except for the two types hard functions, Shekel functions and Griewank function, for which we set it equal to $2n$.

2.3.2 Numerical results

To examine the performance of our algorithms, we tested them on some well known functions [13, 15, 92], which are given in Appendix A. The behavior of these test functions varies; we have functions with some studded local minima such as Goldstein and Price function,

Table 2.1: Percentage of successful trials for SDS, SSA and DSSA

Function	SDS	SSA	DSSA	Function	SDS	SSA	DSSA
RC	100	99	100	$S_{4,5}$	14	61	81
ES	12	68	93	$S_{4,7}$	19	60	84
GP	41	91	100	$S_{4,10}$	15	59	77
B_1	76	100	100	R_5	4	67	100
HM	100	100	100	Z_5	100	87	100
SH	59	57	94	$H_{6,4}$	52	49	92
R_2	12	93	100	GR	36	82	90
Z_2	100	99	100	R_{10}	0	78	100
DJ	100	100	100	Z_{10}	0	66	100
$H_{3,4}$	88	86	100				

functions with many crowded local minima such as Shubert function, functions with a global minimum lying in a very narrow hole such as Easom function, functions with a narrow valley such as Rosenbrock function, and smooth functions such as De Joung function and Zakharov function. For each function we made 100 trials with different starting points. The average number of function evaluations and the average error are related to only successful trials.

First, to demonstrate the effect of hybridizing simulated annealing with SDS to design SSA and DSSA, we show in Table 2.1 the percentage of successful trials. From Table 2.1, we see that the rate of success for SSA is generally better than that for SDS. However, the behavior of SDS for Zakharov function Z_5 is better than that of SSA due to the fixed cooling schedule in SSA for all functions. We note that the behavior of these methods has changed drastically when the dimension n of function Z_n is increased to 10. Moreover, Table 2.1 clearly shows that the behavior of DSSA is the best of the three methods in terms of the rate of success.

In Table 2.2, we show the effect of accelerating the cooling schedule in SSA and applying a local search method on the final results obtained by the accelerated SSA to design DSSA. The results in Table 2.2 reveal that the acceleration procedure successfully affects the rate of success, the average number of function evaluations (Av. f-evals.), and the average error (Av. Error).

To show to what extent DSSA succeed in accelerating SA, we compare its results with other simplex SA method like SIMPSA and NE-SIMPSA [13]. Table 2.3 shows the average

Table 2.2: Results of SSA and DSSA

	Rate of Success		Av. f-evals.		Av. Error	
f	SSA	DSSA	SSA	DSSA	SSA	DSSA
RC	99	100	12225	118	9E-3	4E-7
ES	68	93	4318	1442	4E-3	3E-9
GP	91	100	11238	261	5E-3	4E-9
B_1	100	100	4564	252	7E-3	5E-9
HM	100	100	10157	225	0.01	5E-8
SH	57	94	10237	457	0.1	9E-6
R_2	93	100	7387	306	3E-3	4E-9
Z_2	99	100	5868	186	8E-3	4E-9
DJ	100	100	6743	273	3E-3	5E-9
$H_{3,4}$	86	100	17756	572	0.1	2E-6
$S_{4,5}$	61	81	7856	993	6E-3	2E-6
$S_{4,7}$	60	84	9047	932	0.01	6E-7
$S_{4,10}$	59	77	9062	992	0.01	1E-5
R_5	67	100	11115	2685	0.03	3E-9
Z_5	87	100	11527	914	0.03	5E-9
$H_{6,4}$	49	92	37467	1737	0.02	2E-6
GR	82	90	12208	1830	0.1	5E-9
R_{10}	78	100	22306	16785	0.02	7E-9
Z_{10}	66	100	23883	12501	0.04	7E-9

Table 2.3: Average number of function evaluations in DSSA and other simplex SA methods

Function	DSSA	SIMPSA	NE-SIMPSA
R_2	306	10780	4508
R_4	1682	21177 (99%)	3053 (94%)
CV	1592	22615	3443
DX	6941	52556 (93%)	8613 (94%)

number of function evaluations obtained by each method starting from the same starting point as in [13]. The data for SIMPSA and NE-SIMPSA are taken from [13]. Actually, the reference [13] reports many results for SIMPSA and NE-SIMPSA depending on the search domain but we prefer to make our method more general without any constrains during the search. Moreover, we have chosen the best results obtained by SIMPSA and NE-SIMPSA from Table 2.3 in [13] to make the comparison simpler and fair.

Next we compare the DSSA method with three other metaheuristics methods based on simulated annealing, tabu search, and genetic algorithm. These methods are:

1. Enhanced Continuous Tabu Search (ECTS) [15].

2. Enhanced Simulated Annealing (ESA) [85].

3. Real-value Coding Genetic Algorithm (RCGA) [11].

Table 2.4 shows the average number of function evaluations needed by each algorithm. The results of ECTS, ESA and RCGA are taken from their original references [15, 85, 11]. For all test functions, we use the same condition as that used by ECTS [15] to judge the success of a trial which is given by

$$|f^* - f_{DSSA}| < \epsilon_1 |f^*| + \epsilon_2, \tag{2.3.1}$$

where f_{DSSA} refers to the best function value obtained by DSSA and f^* refers to the exact global minimum. We set $\epsilon_1 = 10^{-4}$ and $\epsilon_2 = 10^{-6}$. The ESA method used the same condition for testing the successful trials with smaller ϵ_1 and ϵ_2. However, for the results marked by ($^{\otimes}$) in Table 2.4, their original corresponding data in Table 2 in [15] and Table I in [85] seem to contain some inconsistencies. Since the authors of [15] used the same condition as (2.3.1) to test the successful trials, the average errors for the functions R_2, R_5 and Z_5 must be less than 10^{-6} because $f^* = 0$ for all these functions. However, the average errors corresponding to these functions are reported to be greater than 10^{-6}. For instance, the average error corresponding to the function R_5 in Table 2 in [15] is 0.08, i.e., there are some trials that did not satisfy the successful trial condition but the authors reported that the rate of success equals 100%. Moreover, the results corresponding to the functions $RC, ES, GP, H_{3,4}$ and $H_{6,4}$ in Table 2 in [15] also contain the same kind of inconsistencies. For the same reasons, the ESA results marked by ($^{\otimes}$) suffer from the same inconsistencies. The comparison given in Table 2.4 shows that DSSA outperforms the others for some functions and has similar behavior for other functions. However, Table 2.5 shows that DSSA generally produces more accurate solutions than the others. It is noteworthy that the efficiency of the simplex method dwindles with dimensionality, which explains the greatest margin of superiority for DSSA on Z_5 while it does not outperform the others on Z_{10}.

2.4 Conclusion

The simulated annealing method usually suffers from slow convergence due to its random nature of movements. Moreover, simulated annealing also suffers from the difficulty in ob-

Table 2.4: Average number of function evaluations in DSSA and other metaheuristics

Function	DSSA	ECTS	ESA	RCGA
RC	118	$245^{\otimes}$	-	490
ES	1442 (93%)	$1284^{\otimes}$	-	642
GP	261	$231^{\otimes}$	$783^{\otimes}$	270
SH	457 (94%)	370	-	946
R_2	306	$480^{\otimes}$	796	596
Z_2	186	195	15820	437
DJ	273	338	-	395
$H_{3,4}$	572	$548^{\otimes}$	$698^{\otimes}$	324
$S_{4,5}$	993 (81%)	825 (75%)	$1137^{\otimes}$ (54%)	1158 (62%)
$S_{4,7}$	932 (84%)	910 (80%)	$1223^{\otimes}$ (54%)	1143 (70%)
$S_{4,10}$	992 (77%)	989 (75%)	$1189^{\otimes}$ (50%)	1235 (58%)
R_5	2685	$2142^{\otimes}$	5364	4150 (60%)
Z_5	914	$2254^{\otimes}$	96799	1115
$H_{6,4}$	1737 (92%)	$1520^{\otimes}$	$2638^{\otimes}$	937
R_{10}	16785	15720 (85%)	$12403^{\otimes}$	8100 (70%)
Z_{10}	12501	4630	$15820^{\otimes}$	2190

Table 2.5: Average errors in function value in DSSA and other metaheuristics

Function	DSSA	ECTS	ESA	RCGA
RC	4E-7	5E-2	-	3E-3
ES	3E-9	1E-2	-	3E-9
GP	4E-9	2E-3	9E-3	1E-9
SH	9E-6	1E-3	-	6E-4
R_2	4E-9	2E-2	-	1E-12
Z_2	4E-9	2E-7	-	1E-10
DJ	5E-9	3E-8	-	6E-4
$H_{3,4}$	2E-6	9E-2	5E-4	7E-3
$S_{4,5}$	2E-6	1E-2	4E-3	1E-3
$S_{4,7}$	6E-7	1E-2	8E-3	1E-4
$S_{4,10}$	1E-5	1E-2	4E-2	4E-3
R_5	3E-9	8E-2	-	1E-1
Z_5	5E-9	4E-6	-	9E-4
$H_{6,4}$	2E-6	5E-2	6E-2	3E-2
R_{10}	7E-9	2E-2	4E-2	1E-1
Z_{10}	7E-9	2E-7	2E-3	3E-3

taining some required accuracy although it may quickly approach the neighborhood of the global minimum. In this chapter, we have focused on the importance of creating direct-search-based logical movements while applying simulated annealing and the importance of accelerating the final stage of simulated annealing by using a faster convergent method. The obtained results demonstrate that these two concepts can be successfully realized by effectively combining direct search methods with simulated annealing. Moreover, the experimental results show that the DSSA method is very efficient and robust.

Chapter 3

Simplex Coding GA for Unconstrained Global Optimization

3.1 Introduction

Genetic algorithms (GAs) are one of the most efficient metaheuristics [34, 68], that have been employed in a wide variety of problems. However, GAs, like other metaheuristics, suffer from the slow convergence that brings about the high computational cost. Recently, several new approaches have been developed to furnish GAs with the ability to simulate the fast convergence of local search methods. Most of these approaches hybridize local search methods with GAs to obtain more efficient methods with relatively faster convergence. This chapter pursues in that direction and proposes a new hybrid method that combines GA with Nelder-Mead method [72] to deal with the unconstrained optimization problem

$$\min_{x \epsilon R^n} \ f(x), \tag{3.1.1}$$

where f is a generally nonconvex, real valued function defined on R^n. In the combined method, called the simplex coding genetic algorithm (SCGA), we consider the members of the population to be simplices, i.e., each chromosome is a simplex and the gene is a vertex of this simplex. Selection, crossover and mutation procedures are used to improve the initial population. Moreover, Nelder-Mead method is applied to improve the population in the initial stage and every intermediate stage when new children are generated. In the SCGA, we use the linear ranking selection scheme [9] to choose some fit parents to be mated. Then,

using a new scheme of a multi-parents crossover, new children are reproduced and a few of them are mutated. Applying Kelley's modification [51, 52] of Nelder-Mead method on the best point visited is the final stage in the SCGA to accelerate the search and to improve this best point.

There have been some attempts to utilize the idea of hybridizing local search methods with GA. Simple hybrid methods use the GA or local search methods to generate the points for the new population and then apply the other technique to improve this new population [35, 96]. Other hybrid methods do some modifications in the GA operations; selection, crossover and mutation using local search methods [71, 80, 94, 95]. However, the method proposed in this chapter is different from those hybrid methods as we will see in the next section. The next section briefly reviews some hybrid GA methods that use Nelder-Mead method. The description of the proposed method is given in Section 3.3. Section 3.4 discusses the experimental results along with the initialization of some parameters and the setting of the control parameters of the proposed method. The conclusion of the contribution of this chapter follows the experimental results and makes up Section 3.5.

3.2 Simplex-Based Genetic Algorithms

In this section, we review some earlier methods that hybridize GAs and simplex methods. The Nelder-Mead method is the most popular simplex-type method that has been used to design a hybrid simplex-based GA method. There have been several attempts to hybridize GA with simplex-based direct search methods. Remarkable features underlying these hybrid methods are global exploration and parallelism in GA, and local exploitation in direct search methods. Moreover, both GA and direct search methods only use the function values rather than derivatives, which makes those hybrid methods applicable to a broad class of problems. In the following, we briefly summarize some of the hybrid simplex-based GA methods.

Renders and Bersini method [80]. In this method the population is divided into λ groups of $n+1$ chromosomes. Then, one of the following operations is applied to each group with some predetermined probabilities to reproduce exactly one child.

- *Discrete crossover.* Each gene in a child can be chosen from the corresponding gene in a parent which is randomly chosen from the group. This child replaces the worst parent in this group.
- *Average crossover.* The average of all $n+1$ parents in the group replaces the worst parent in this group.

- *Simplex crossover.* Apply the Nelder-Mead method with slight modification to reproduce a new point.

The algorithm terminates if some convergence criterion is reached.

Yang and Douglas method [94]. M points are selected randomly from the search space to form the initial population. GA's reproduction schemes (selection, crossover, and mutation) are used to generate k $(0 < k < M)$ children. The rest of the offspring will be generated by repeating the following procedure $M - k$ times. Using some selection scheme, construct a subcommunity of S points from the M old points. Try to get a better child by applying a simplex method. Otherwise, a child is generated randomly within the search space. If the best point of the new generation is not better than the best one of the old generation, then replace the worst point of the new generation by the best point of the old generation. Moreover, some comparisons are made between the old generation and the new one to copy some of the best points in the old generation into the new one. The algorithm terminates if either a predetermined iteration number is reached or an acceptable objective function value is obtained.

Yen, Liao, Lee, and Randolph method [95]. This simplex GA hybrid method uses a modification of the Nelder-Mead method called the concurrent simplex method. The initial population consists of M chromosomes and the concurrent simplex method is applied to the top S $(n < S < M)$ chromosomes in the population to produce $S - n$ children. The top n chromosomes are copied to the next generation. The GA's reproduction operations, crossover and mutation, are used to generate the remaining $M - S$ chromosomes. The algorithm terminates when it satisfies a convergence criterion or reaches a predetermined maximum number of fitness evaluations.

Musil, Wilmut, and Chapman method [71]. The initial population consists of M chromosomes generated at random. The cycle starts by selecting $n + 1$ random pairs of parents from the population. The binary operations (crossover and mutation) are applied on the parents to reproduce children. One child is selected from each of the $n + 1$ pairs of children and this results in $n + 1$ new children. The Nelder-Mead method runs for k iterations starting with the simplex that consist of these $n+1$ children. The point that gives the lowest objective function value obtained by Nelder-Mead iterations replaces the one with the highest objective function value in the population. The cycle is terminated when the parameters in the population have converged. At this point, other Nelder-Mead iterations start with the chromosome with the lowest objective function value in the population to refine this chromosome and to get the solution.

Our hybrid method SCGA presented in the next section is different from these hybrid

methods in many aspects. One of the main differences lies in the coding representation. We use a simplex coding in which the chromosome is a simplex and its genes are the vertices of this simplex. It is expected that using this coding type and applying some iteration of Nelder-Mead method starting from each chromosome in the initial population and from each child chromosome will increase the local exploitation and will improve these chromosomes. The other main difference consists in the crossover operation. We introduce a new kind of multi-parents crossover that gives the chance for more than two parents to cooperate in reproducing children and exploring the region around these parents.

3.3 Description of SCGA

In this section, we describe the proposed method SCGA. The SCGA uses the main functions of the GA; selection, crossover and mutation, on a population of simplices to encourage the exploration process. Moreover, the SCGA tries to improve the initial members and new children by applying a local search method to enhance the exploitation process. This kind of exploration-exploitation procedure is sometimes called "Memetic Algorithm", see [70]. Finally, the SCGA applies an effective local search method on the best point reached by the previous exploration-exploitation procedure. The purpose of this local search is to accelerate the final stages of the GA procedure. This strategy is expected to be effective because the GA has a difficulty in obtaining some required accuracy although the GA may quickly approach the neighborhood of the global minimum.

3.3.1 Initialization

The SCGA starts with the following initialization procedure:

1. Generate the initial population P_0 that consists of M chromosomes (simplices), i.e.,

$$P_0 = \left\{ S^j : S^j = \left\{ x^{j,i} \right\}_{i=1}^{n+1};\ x^{j,i} \in R^n,\ j = 1, \ldots, M \right\}.$$

2. Order the vertices of each simplex S^j, $j = 1, 2, \ldots, M$, so that

$$f(x^{j,1}) \leq f(x^{j,2}) \leq \cdots \leq f(x^{j,n+1}). \tag{3.3.1}$$

3. Apply a small number of iterations of the Nelder-Mead method with each S^j as an initial simplex to improve the chromosomes in the initial population P_0.

4. Order the simplices $S^j = \{x^{j,i}\}_{i=1}^{n+1},\ j = 1, \ldots, M$ in the improved population P_0 so that

$$f(x^{1,1}) \leq f(x^{2,1}) \leq \cdots \leq f(x^{M,1}). \tag{3.3.2}$$

3.3.2 GA loop

Repeat the following procedures; selection, crossover and mutation, and reduction of the population, described below until the stopping conditions are not satisfied.

Selection

We describe how we select the set $Q \subseteq P$ of the members that will be given the chance to be mated from the current population P. For each generation, the size of Q is the same as that of P but more fit members in P are chosen with higher probability to be included in Q. We use Baker's scheme called "linear ranking selection" [9] to select the new members in Q. In this scheme, the chromosomes $S^j \in P$, $j = 1, 2, \ldots, M$, are sorted in the order of raw fitness as in (3.3.2), and then the probability of including a copy of chromosome S^j into the set Q is calculated by

$$p_s(S^j) = \frac{1}{M}\left(\eta_{\max} - (\eta_{\max} - \eta_{\min})\,\frac{j-1}{M-1}\right),$$

where $\eta_{\min} = 2 - \eta_{\max}$ and $1 \leq \eta_{\max} \leq 2$. Using these probabilities, the population is mapped onto a roulette wheel, where each chromosome S^j is represented by a space that proportionally corresponds to $p_s\left(S^j\right)$. Chromosomes in the set Q are chosen by repeatedly spinning the roulette wheel until all positions in Q are filled.

Crossover and mutation

Choose a random number from the unit interval $(0, 1)$ for each chromosome in Q. If this number is less than the predetermined crossover probability p_c, then this chromosome is chosen as a parent. Repeat the following steps until all parents are mating.

1. Select a number n_c from the set $\{2, \ldots, n+1\}$ randomly to determine the number of parents chosen to be mated together.

2. Compute new children $C^i = \left\{x_c^{i,k}\right\}_{k=1}^{n+1},\ i = 1, \ldots, n_c$ by

$$x_c^{i,k} = \overline{x}^k + d\, r^i,\ k = 1, \ldots, n+1, \tag{3.3.3}$$

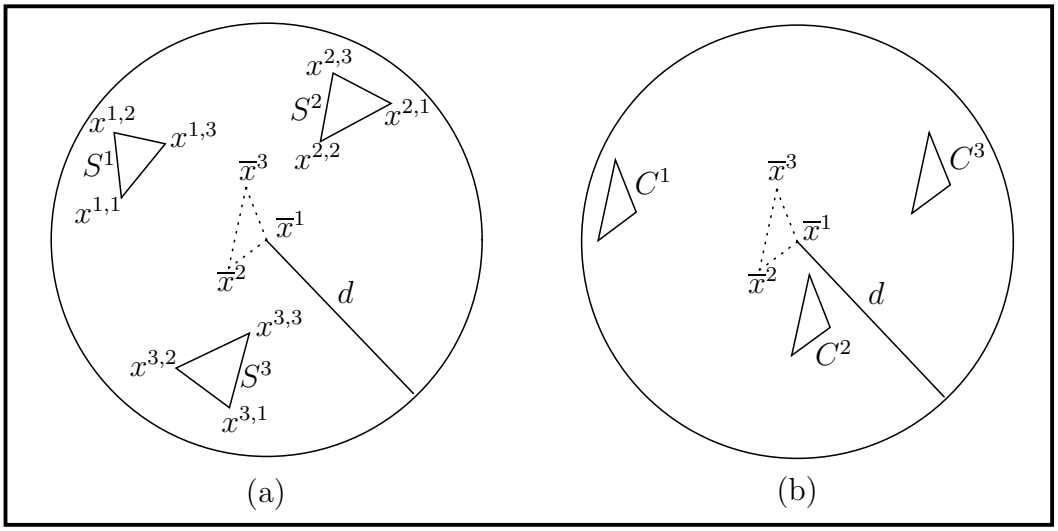

Figure 3.1: An example of SCGA crossover in two dimensions.

where $r^i,\ i = 1, \ldots, n_c$, are random vectors of length less than 1, d is the maximum distance between pairs of parents and $\overline{x}^k$ is the average of the kth vertices of all parents, i.e.,

$$\overline{x}^k = \frac{1}{n_c} \sum_{i=1}^{n_c} x^{i,k},\ k = 1, \ldots, n+1. \tag{3.3.4}$$

Figure 3.1 shows an example of crossover in two dimensions. In Figure 3.1(a), we use Equations (3.3.4) to compute the dotted simplex whose vertices are the average of the vertices of the parents S^1, S^2 and S^3. By using Equations (3.3.3), we move this dotted simplex randomly inside the circle to create the children C^1, C^2 and C^3, as in Figure 3.1(b).

3. Choose a random number from the unit interval $(0, 1)$ for each child $C^i,\ i = 1, \ldots, n_c$. If this number is less than the predetermined mutation probability p_m, then this child is mutated. Let I_m be the index set of those children who are mutated.

4. Apply the following procedure for each child $C^i = \left\{x_c^{i,k}\right\}_{k=1}^{n+1},\ i \in I_m$. Select a number n_i from the set $\{1, 2, \ldots, n+1\}$ randomly to determine the vertex that is reflected as a mutation. Compute the mutated child $\widetilde{C}^i = \left\{x_m^{i,k}\right\}_{k=1}^{n+1}$ by

$$\begin{aligned} x_m^{i,k} &= x_c^{i,k},\ k = 1, \ldots, n_i - 1, n_i + 1, \ldots, n+1, \\ x_m^{i,n_i} &= \overline{x} + u\left(\overline{x} - x_c^{i,n_i}\right), \end{aligned}$$

where u is a random number in the interval $[0.5, 1.5]$ and $\overline{x}$ is the average of vectors $x_c^{i,1}, \ldots, x_c^{i,n_i-1}, x_c^{i,n_i+1}, \ldots, x_c^{i,n+1}$. Replace the child C^i by the mutated one $\widetilde{C}^i$. Figure

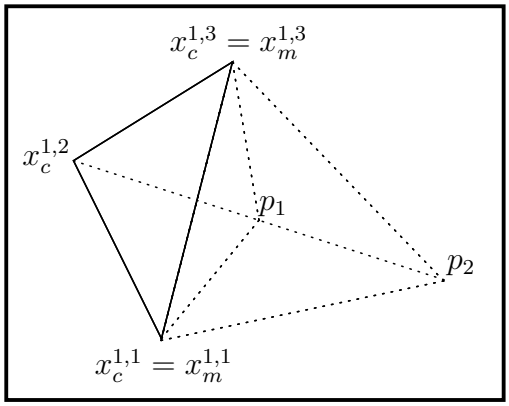

Figure 3.2: An example of SCGA mutation in two dimensions.

3.2 shows an example of mutation in two dimensions, where the mutated simplex consists of the vertices $x_m^{1,1}$, $x_m^{1,2}$ and $x_m^{1,3}$, where the vertex $x_m^{1,2}$ is randomly chosen on the line segment $\overline{p_1 p_2}$.

5. Apply a small number of iterations of the Nelder-Mead method with each child $C^i,\ i = 1, \ldots, n_c$ as an initial simplex to improve the chromosomes.

6. The population in the next generation consists of the M best ones from the set $P \cup \{C^i\}_{i=1}^{n_c}$. Re-order the chromosomes in the new population so that (3.3.1) and (3.3.2) hold.

Reduction of the population

After every predetermined number of generations, remove some of the worst members in the population P.

3.3.3 Acceleration in the final stage

From the best point obtained in GA loop, construct a small simplex. Then, apply Kelley's modification [51, 52] of the Nelder-Mead method on this simplex to obtain the final solution.

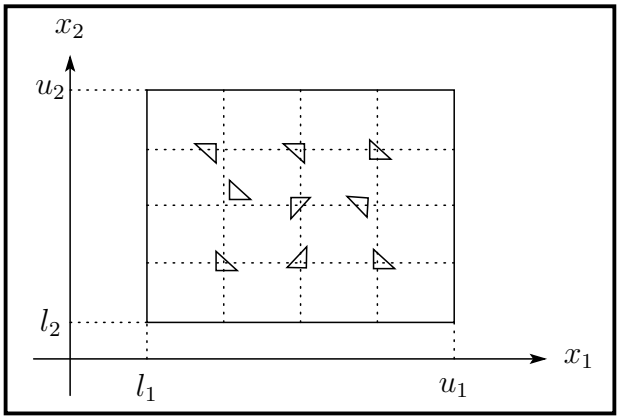

Figure 3.3: Initial population in two dimensions.

3.4 Experimental Results

3.4.1 Parameter setting

In this subsection, we specify suggested values of the initial and control parameters.

Generating the initial population

Let $[L, U] = \{x \in R^n : l_i \leq x_i \leq u_i,\ i = 1, 2, \ldots, n\}$ be the domain in which the initial points are chosen and let this domain be divided into an equal distance grid. The population consists of simplices distributed on this grid and in each coordinate direction there are μ simplices, i.e., the size M of population equals μ^n. We distribute the simplices in such a way that each simplex is put in a neighborhood of one of the knots in the grid. We consider all possible positions of simplices if $n = 2$, and some best positions of them if $3 \leq n \leq 10$. However, for $n > 10$, we employ another procedure for choosing the initial simplices as will be discussed later. From each one of these main vertices we construct a right-angled simplex by taking a step in each coordinate direction. This step size is called edge length. Figure 3.3 shows an example of the distribution of the population in two dimensions. The values of the parameters used in generating the initial population are given as follows.

1. The number μ of simplices per coordinate direction is varied from 2 to 5 according the estimated density of the local minima of the test function and the number n of variables.

2. The edge length is set equal to $\max_{1\leq i\leq n}(u_i - l_i)/10$.

3. The number of Nelder-Mead iterations in the local search for the initial population is set equal to 2.

In the case of $n > 10$, the main vertices in the initial population are generated in a different way. First, we choose a random point $x^{1,1} \in [L, U]$. Then we generate the other main vertices $x^{j,1},\ j = 2, \ldots, M$, by using the following procedure:

1. Generate $x^{j,1}$.

2. If $\min_{1\leq k\leq j-1} \max_{1\leq i\leq n} |x_i^{j,1} - x_i^{k,1}|/(u_i - l_i) \geq h$ for some prescribed $h \in (0, 1)$, then accept this vertex $x^{j,1}$. Otherwise, return to step 1.

We set the probability of accepting the new main vertex equal to 0.7. From these main vertices, we construct the simplices of the initial population as in the case of $n \leq 10$.

GA loop parameters

The steps of the GA loop have been described in the previous section. Here we specify the values of the parameters used in this loop.

1. The control parameter $\eta_{\max}$ in the selection procedure is chosen to be 1.1 according to the original setting in [9].

2. The crossover probability p_c and the mutation probability p_m are set equal to 0.6 and 0.1, respectively.

3. The number of Nelder-Mead iterations in the local search for the new children is fixed at 2.

4. At every $3n$ generations, we remove the n worst chromosomes from the population unless the number of its chromosomes is less than $2n$.

Termination criteria

The SCGA is terminated when one of the following termination conditions is satisfied.

1. The function values at all vertices of the simplex that contains the best point become close to each other, i.e.,
$$f(x^{1,n+1}) - f(x^{1,1}) \leq \epsilon,$$
where the tolerance ϵ is a small positive number and set equal to 10^{-8}.

2. The number of generations exceeds the predetermined number that is set equal to $\min(10n, 100)$.

3.4.2 Numerical results

The performance of the SCGA was tested on a number of well known functions [11, 12, 37, 94, 95], which are given in Appendix A. The behavior of these test functions varies to cover many kinds of difficulties that face unconstrained global optimization problems. For each function we made 100 trials with different initial populations. To judge the success of a trial, we used the condition

$$|f^* - \widehat{f}| < \epsilon_1 |f^*| + \epsilon_2, \tag{3.4.1}$$

where $\widehat{f}$ refers to the best function value obtained by SCGA, f^* refers to the known exact global minimum, and ϵ_1 and ϵ_2 are small positive numbers. We set ϵ_1 and ϵ_2 equal to 10^{-4} and 10^{-6}, respectively, when $n \leq 10$, but for $n > 10$ we relax this test condition by increasing the value of ϵ_2 to 10^{-4}. The results are shown in Table 3.1, where the average number of function evaluations (Av. f-evals.) and the average error (Av. Error) are related to only successful trials. Table 3.1 shows that the SCGA reached the global minima in a very good success rate for the majority of the tested functions. Moreover, the numbers of function evaluations and the average errors show the efficiency of the method.

In Table 3.2, we compare the results of the SCGA with those of three other metaheuristic methods. These methods are:

1. Real-value Coding Genetic Algorithm (RCGA) [11].

2. Continuous Genetic Algorithm (CGA) [16].

3. Direct Search Simulated Annealing (DSSA) proposed in Chapter 2.

Table 3.1: Results of SCGA

f	Rate of success	Av. f-evals.	Av. Error
RC	100	173	3.62e-07
ES	100	715	4.97e-09
GP	100	191	4.81e-09
HM	100	176	5.23e-08
SH	98	742	8.83e-06
MZ	100	179	3.40e-06
B_1	99	460	5.11e-09
B_2	99	471	5.43e-09
B_3	100	468	5.14e-09
R_2	100	222	4.60e-09
Z_2	100	170	4.68e-09
DJ	100	187	5.12e-09
$H_{3,4}$	100	201	2.14e-06
$S_{4,5}$	79	1086	3.28e-07
$S_{4,7}$	81	1087	4.06e-05
$S_{4,10}$	84	1068	9.81e-06
R_5	90	3629	5.88e-09
Z_5	100	998	7.10e-09
$H_{6,4}$	99	989	2.00e-06
GR	100	906	8.46e-09
R_{10}	90	6340	1.85e-08
Z_{10}	100	1829	1.76e-08
R_{20}	90	33134	7.59e-05
Z_{20}	100	33106	5.79e-07

Table 3.2: Average number of function evaluations in SCGA and other metaheuristics

Function	SCGA	RCGA [11]	CGA$^{\otimes}$ [16]	DSSA
RC	173	490	620	118
ES	715	642	1504	1442 (93%)
GP	191	270	410	261
HM	176	-	-	225
SH	742 (98%)	946	575	457 (94%)
MZ	179	452	-	-
B_1	460 (99%)	-	430	252
B_2	471 (99%)	493	-	-
R_2	222	596	960	306
Z_2	170	437	620	186
DJ	187	395	750	273
$H_{3,4}$	201	342	582	572
$S_{4,5}$	1086 (79%)	1158 (62%)	610 (76%)	993 (81%)
$S_{4,7}$	1087 (81%)	1143 (70%)	680 (83%)	932 (84%)
$S_{4,10}$	1068 (84%)	1235 (58%)	650 (81%)	992 (77%)
R_5	3629 (90%)	4150 (60%)	3990	2685
Z_5	998	1115	1350	914
$H_{6,4}$	989 (99%)	973	970	1737 (92%)
GR	906	-	-	1830 (90%)
R_{10}	6340 (90%)	8100 (70%)	21563 (80%)	16785
Z_{10}	1829	2190	6991	12501

The figures for RCGA and CGA methods in Table 3.2 are taken from the original references. For those results of the CGA which are marked by ($^{\otimes}$) in Table 3.2, their original corresponding data in Table 1 in [16] seem to contain some inconsistencies. In fact, since the same condition as (3.4.1) is used in CGA [16] to test the successful trials, the average errors for the tested functions must be less than the right-hand side of (3.4.1) for all these functions. However, the average errors corresponding to the tested functions in [16] are reported to be greater than the right-hand side of (3.4.1). The comparison given in Table 3.2 shows the SCGA outperforms the others for many of those functions.

Next, we try to compare SCGA with some of the other simplex-based GA methods described in Section 3.2. Actually, for many reasons, it is not so easy to make clear comparisons between SCGA and other simplex-based GA methods of [71, 80, 94, 95]. In fact, some of these hybrid methods such as [71, 95] are concentrated on a certain complicated specific

Table 3.3: The results for F_1 function

Function evaluations	SCGA	Simplex GA [94]
Average	351	660
Min	259	32
Max	452	9538

problem. Moreover, for some of these methods, computational experiments reported in their original references do not show much helpful information for comparison. For instance, the successful trial test is not mentioned in [94, 95] and the number of the test problems is very small in [71, 80, 94]. Nevertheless, in Tables 3.3 and 3.4, we give the available comparisons between SCGA and the other Simplex GA methods of [80, 94, 95]. First, we compare SCGA with the Simplex GA [94] using two functions F_1 ($n = 2$) and F_2 ($n = 10$), see Appendix A. The results for F_1 are shown in Table 3.3 and the results for both SCGA and the Simplex GA [94] are taken over 100 trials. It is seen that for this function, SCGA outperforms the Simplex GA [94] with regard to the average function evaluations. For the other function F_2, many results for the Simplex GA are reported in [94] and the best of them is 0.0002 for the best function value with 6400 function evaluations. On the other hand, the results of SCGA for this function are slightly worse, that is, 0.0008 for the best function value with 8127 function evaluations. However, there is another function with $n = 10$ studied in [94] for which the Simplex GA [94] needed to generate 640 generations to obtain the accuracy 10^{-3}, whereas SCGA needed only 60 generation to obtain the accuracy 10^{-9}.

SCGA is also compared with the Simplex GA methods of [80, 95] using De Joung F5 function, see Appendix A. All these methods have the same rate of success (100%), but the Simplex GA [80] required a large number of function evaluations, as shown in Table 3.4. We note that the results for the Simplex GA methods of [80, 95], which are cited from [95], are the average taken over 10 trials. However, the reference [95] does not give the condition used to judge the success of trials for both of the Simplex GA methods of [80, 95], whereas the results for SCGA are taken over 100 trials and we use condition (3.4.1) with $\epsilon_1 = 10^{-4}$ and $\epsilon_2 = 10^{-6}$ to judge the success of trials. It is noteworthy that the average error obtained by SCGA for De Joung function F5 is 1.6×10^{-7}.

Table 3.4: The results for De Joung F5 function

Method	Average number of function evaluations
SCGA	1570
Simplex GA [80]	14924
Simplex GA [95]	1695

3.5 Conclusion

In this chapter, we have introduced a simplex coding genetic algorithm that uses a set of simplices as the population. Applying the Nelder-Mead local search method on these simplices in addition to the ordinary GA operations such as selection, crossover and mutation enhances the exploration process and accelerates the convergence of the GA. We also have introduced a new kind of multi-parents crossover that gives more than two parents the chance to cooperate in reproducing children and exploring the region around these parents. Moreover, using a local search method again in the final stage helps the GA in obtaining good accuracy quickly. Finally, the computational results show that the SCGA works successfully on some well known test functions.

Chapter 4

Heuristic Pattern Search SA for Unconstrained Global Optimization

4.1 Introduction

Simulated annealing (SA) [1, 53, 57, 58] is one of the most effective metaheuristics not only for combinatorial optimization but also for continuous global optimization. However, SA suffers from slow convergence and also it may wander around the optimal solution if high accuracy is needed. In continuous optimization, combining SA with direct search methods is a practical remedy to overcome the slow convergence of SA as shown previously in Chapter 2. In this chapter, we present a new hybrid method that combines SA with a new pattern search method to deal with the unconstrained optimization problem

$$\min_{x \epsilon R^n} \ f(x), \tag{4.1.1}$$

where f is a generally nonconvex, real valued function defined on R^n.

We will make use of two new ideas to form the main parts of the hybrid algorithm. We first introduce a derivative-free heuristic method to produce an approximate descent direction at the current solution, which we call the Approximate Descent Direction (ADD) method. Some preliminary numerical results show that the ADD method has a high ability to obtain a descent direction. Next, we use the ADD method to design a new PS method called the Heuristic Pattern Search (HPS) method. In the HPS method, the ADD method is recalled

to obtain an approximate descent direction v at the current iterate. If no improvement is obtained along the vector v, then we use v to prune the set of pattern search directions to generate other exploratory moves. Finally, we hybridize SA and HPS to construct a global search method, called the Simulated Annealing Heuristic Pattern Search (SAHPS) method. The SAHPS tries to get better movements through the SA acceptance procedure or by using the HPS procedure. More specifically, we first introduce a new exploring neighborhood search to generate a number of SA trial points. If some of these trail points can be accepted by the SA acceptance procedure, this means the search can go further and there is no need to use a local search method. Otherwise, we apply some iterations of the HPS method to generate more local exploratory trial points. In the final stage of the search, we apply a direct search method to refine the best solution obtained so far. Numerical results with 19 well-known test functions indicate that the SAHPS exhibits a very promising performance to obtain global minima of multimodal functions.

This chapter is organized as follows. We introduce the ADD and the HPS methods with some numerical results to show their performances in Section 4.2 and Section 4.3, respectively. The description of the main SAHPS method is given in Section 4.4. In Section 4.5, we discuss the experimental results along with the initialization of some parameters and the setting of the control parameters of the SAHPS method. Finally, the conclusion of the contribution of this chapter makes up Section 4.6.

4.2 Approximate Descent Direction

In this section, we present the ADD method in which we use m close exploring points to generate an approximate descent direction. Given a point $p \in R^n$, we want to obtain an approximate descent direction $v \in R^n$ of f at p. We randomly generate m points $\{y_i\}_{i=1}^m$ close to p and compute the direction v at p as follows:

$$v = \sum_{i=1}^{m} w_i e_i, \tag{4.2.1}$$

where

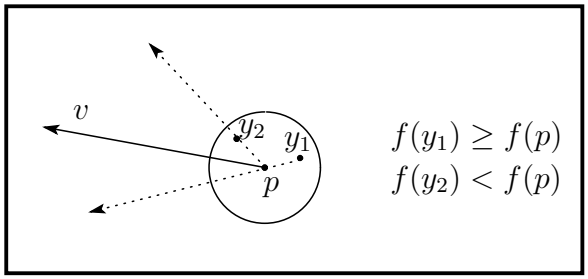

Figure 4.1: An ADD example in R^2.

$$\begin{aligned} w_i &= \frac{\Delta f_i}{\sum_{j=1}^{m} |\Delta f_j|}, \quad i = 1, 2, \ldots, m, \\ e_i &= -\frac{(y_i - p)}{\|y_i - p\|}, \quad i = 1, 2, \ldots, m, \\ \Delta f_i &= f(y_i) - f(p), \quad i = 1, 2, \ldots, m. \end{aligned} \tag{4.2.2}$$

By means of (4.2.1), the direction v is composed toward the vectors $-sign\left(\Delta f_i\right)\left(y_i - p\right)$ with weights proportional to $\left|\Delta f_i\right|$, $i = 1, 2, \ldots, m$. Figure 4.1 shows an example of composing an ADD in two dimensions. Given a point $p \in R^2$, the ADD v is composed in Figure 4.1 toward

- the vector $-\left(y_1 - p\right)$, since the inequality $f\left(y_1\right) \geq f\left(p\right)$ suggests that the function value is not likely to decrease along the direction $y_1 - p$, and

- the vector $y_2 - p$, since the inequality $f\left(y_2\right) < f\left(p\right)$ suggests that the function value is likely to decrease along the direction $y_2 - p$.

We can show some theoretical results concerning the descent property of the direction v in the following two special cases.

The linear case. If f is a linear function, i.e., $f\left(x\right) = c^T x + b$, $c \in R^n, b \in R$, then the

vector v in (4.2.1) can be written as

$$\begin{aligned} v &= \frac{-1}{\sum_{j=1}^{m}|\Delta f_j|}\sum_{i=1}^{m}\Delta f_i\frac{(y_i-p)}{\|y_i-p\|} \\ &= \frac{-1}{\sum_{j=1}^{m}|\Delta f_j|}\sum_{i=1}^{m}c^T(y_i-p)\frac{(y_i-p)}{\|y_i-p\|} \\ &= \frac{-1}{\sum_{j=1}^{m}|\Delta f_j|}\left(\sum_{i=1}^{m}\frac{(y_i-p)(y_i-p)^T}{\|y_i-p\|}\right)c \\ &= -\gamma Ac, \end{aligned}$$

where $\gamma = 1/\sum_{j=1}^{m}|\Delta f_j|$ and $A = \sum_{i=1}^{m}(y_i-p)(y_i-p)^T/\|y_i-p\|$. Note that matrix A is positive semidefinite, since $x^TAx = \sum_{i=1}^{m}\left((y_i-p)^Tx\right)^2/\|y_i-p\| \geq 0$ for any $x \in R^n$. Therefore, it holds that $\nabla f(p)^T v = -\gamma c^TAc \leq 0$, i.e., v is a descent direction.

The nonlinear case. If f is a differentiable nonlinear function, we can approximate f around point p as $f(x) \cong f(p) + \nabla f(p)^T(x-p), \forall x \in N(p)$, where $N(p)$ is a small neighborhood of p. Therefore, if points $y_i,\ i = 1,\ldots,m$, are chosen from the neighborhood $N(p)$, then the vector v in (4.2.1) can be represented approximately as

$$\begin{aligned} v &= \frac{-1}{\sum_{j=1}^{m}|\Delta f_j|}\sum_{i=1}^{m}\Delta f_i\frac{(y_i-p)}{\|y_i-p\|} \\ &\cong \frac{-1}{\sum_{j=1}^{m}|\Delta f_j|}\sum_{i=1}^{m}\nabla f(p)^T(y_i-p)\frac{(y_i-p)}{\|y_i-p\|} \\ &= \frac{-1}{\sum_{j=1}^{m}|\Delta f_j|}\left(\sum_{i=1}^{m}\frac{(y_i-p)(y_i-p)^T}{\|y_i-p\|}\right)\nabla f(p) \qquad (4.2.3) \\ &= -\gamma A\nabla f(p), \qquad (4.2.4) \end{aligned}$$

where A and γ are defined as before. Since A is positive semidefinite, we obtain $\nabla f(p)^T v \cong -\gamma\nabla f(p)^T A\nabla f(p) \leq 0$, i.e., v is expected to be a descent direction.

Remark 4.2.1. *The vector $-\nabla f(p)$, which is referred to as the steepest descent direction of f at p, provides the direction along which the function f decreases most rapidly. Since our aim is to minimize f, it is therefore plausible to try to obtain a direction v that imitates $-\nabla f(p)$. Actually, we can show that the vector v in (4.2.1) can simulate the steepest descent direction $-\nabla f(p)$ under some conditions. Specifically, the vector v becomes approximately*

proportional to $-\nabla f(p)$ *by setting* $m = n$ *and choosing the points* $\{y_i\}_{i=1}^n$ *so as to meet the following conditions:*

- *The points* $\{y_i\}_{i=1}^n$ *are in equal distance from* p*, i.e.,* $\|y_i - p\| = \epsilon,\ i = 1, 2, \ldots, n$*, for some* $\epsilon > 0$*;*
- *the vectors* $\{(y_i - p)\}_{i=1}^n$ *are orthogonal to each other.*

In fact, by letting $u_i = (y_i - p) / \|y_i - p\|$ *for* $i = 1, \ldots, n$*, we may rewrite the formula (4.2.3) as follows:*

$$\begin{aligned} v &\cong \frac{-\epsilon}{\sum_{j=1}^n |\Delta f_j|} \left(\sum_{i=1}^n u_i u_i^T \right) \nabla f(p) \\ &= \frac{-\epsilon}{\sum_{j=1}^n |\Delta f_j|} Q\ \nabla f(p), \end{aligned}$$

where $Q = \sum_{i=1}^n u_i u_i^T$*. Since* $Qu_i = u_i, i = 1, \ldots, n$*, we can readily see* $Q = I_n$*, and this shows that* v *is approximately proportional to* $-\nabla f(p)$*. This result provides a controlled way to generate the exploring points* $\{y_i\}_{i=1}^m$ *rather than a complete random choice of them. Both of these two ways of generating the points* $\{y_i\}_{i=1}^m$ *are tested numerically at the end of this section.*

Remark 4.2.2. *In general, it is not easy to know how small the neighborhood* $N(p)$ *should be in order to ensure the validity of approximation (4.2.4). Let* $N(p) = \{x : \|x - p\| \le \epsilon\}$ *and* $M = \sup\{\nabla^2 f(\zeta) : \zeta \in N(p)\}$*. Then we have for any* $x \in N(p)$

$$\left| f(x) - f(p) - \nabla f(p)^T (x - p) \right| = \left| \frac{1}{2} (x - p)^T \nabla^2 f(p + \theta (x - p))(x - p) \right| \le \frac{1}{2} M \epsilon^2$$

where $\theta \in (0, 1)$*. This estimate may suggest a proper choice of radius* ϵ *of the neighborhood. However, a priori knowledge of* M *is not available except for some special cases. Our numerical experiments reported below suggest that the choice* $\epsilon = 10^{-3}$ *practically works well.*

The previous theoretical analysis uses an approximation of f in the nonlinear case. Here we give some numerical results to show the effectiveness of the ADD method in obtaining a descent direction. We test this procedure using Easom (*ES*), Goldstein and Price (*GP*), Griewank (*GR*) and Rosenbrock ($R_n, n = 2, 4, 10, 20, 50$) functions, as shown in Table 4.1. See Appendix A for the analytical formulae and search domains for these test functions. For each test function, three different test points $p_j, j = 1, 2, 3$, are randomly chosen from its search domains. In addition, three test points $p_j, j = 4, 5, 6$, are chosen to be close to the global minimum x^* for each test function such that $p_4 = x^* - 0.1e$, $p_5 = x^* - 0.01e$ and $p_6 = x^* - 0.001e$, where $e \in R^n$ is the vector of ones. An approximate descent direction v is computed 100 times for each point using different exploring points $\{y_i\}_{i=1}^m$ in each trial. The success rate for obtaining a descent direction in these 100 trials are reported in Table 4.1. The following two methods are used to generate the exploring points $\{y_i\}_{i=1}^m$ close to each point $p = p_j, j = 1, \ldots, 6$:

1. *Random:* Let $m = 2$ and choose points $\{y_i\}_{i=1}^2$ randomly from the neighborhood $N(p, \epsilon) = \{x \in R^n : \|p - x\| \leq \epsilon\}$.

2. *Orthogonal:* Let $m = n$ and choose points $\{y_i\}_{i=1}^n$ such that $\{(y_i - p)\}_{i=1}^n$ are parallel to the coordinate axes and $\|y_i - p\| = \epsilon,\ i = 1, \ldots, n$, for some $\epsilon > 0$.

As to the neighborhood radius ϵ, smaller value of ϵ is expected to yield higher possibility of obtaining a descent direction. To examine how small ϵ is enough to achieve this goal, we have tested three values of ϵ, which are 10^{-1}, 10^{-3} and 10^{-5}. If two percentages are reported in the same space in Table 4.1, the first one is related to *Random* and the second one is related to *Orthogonal*. If only one percentage is reported, this means both of them have this percentage.

The results in Table 4.1 show that using the neighborhood radius $\epsilon = 10^{-3}$ or 10^{-5} is very effective in obtaining a descent direction even in a vicinity of the global minimum. Moreover, there is no significant difference between the results obtained using these two values of ϵ. It is noteworthy that although the *Random* method uses only two random exploring points, it succeeds to obtain a descent direction with a high rate even for higher dimensional functions.

Table 4.1: Success rates of obtaining descent direction for the test functions

f	ϵ	$p_1(\%)$	$p_2(\%)$	$p_3(\%)$	$p_4(\%)$	$p_5(\%)$	$p_6(\%)$
	10^{-1}	100	100	100	100	55/49	45/46
R_2	10^{-3}	100	100	100	100	100	100
	10^{-5}	100	100	100	100	100	100
	10^{-1}	100	100	100	96/100	54/63	42/35
R_4	10^{-3}	100	100	100	100	100	96/100
	10^{-5}	100	100	100	100	100	100
	10^{-1}	100	100	100	94/100	55/47	51/55
R_{10}	10^{-3}	100	100	100	100	100	95/100
	10^{-5}	100	100	100	100	100	100
	10^{-1}	100	100	100	95/100	52/52	57/38
R_{20}	10^{-3}	100	100	100	100	99/100	98/100
	10^{-5}	100	100	100	100	100	100
	10^{-1}	100	100	100	92/100	56/57	48/31
R_{50}	10^{-3}	100	100	100	100	100	97/100
	10^{-5}	100	100	100	100	100	100
	10^{-1}	100	100	100	99/100	90/70	58/76
ES	10^{-3}	100	100	100	100	100	99/100
	10^{-5}	100	100	100	100	100	100
	10^{-1}	100	100	87/48	100	52/45	52/48
GP	10^{-3}	100	100	100	100	100	99/100
	10^{-5}	100	100	100	100	100	100
	10^{-1}	93/100	96/100	87/100	89/100	80/65	51/48
GR	10^{-3}	94/100	96/100	83/100	92/100	91/100	89/100
	10^{-5}	95/100	97/100	89/100	95/100	87/100	90/100

4.3 Heuristic Pattern Search

In this section, we describe the details of the new pattern search method HPS. At the iteration k with iterate $x_k \in R^n$, the HPS uses the ADD method to generate a direction v at x_k. If we could obtain a better movement along direction v with a certain step size, then we proceed to the next iteration by updating the current iterate. Otherwise, the HPS, like conventional pattern search (PS) algorithms [88], uses a finite set D of positive spanning directions in R^n to generate a mesh of points. To avoid searching randomly in all these direction, we prune the positive spanning direction set D, by using a control parameter $\beta \in (-1, 1)$, to select only those directions which lie within the angle $\cos^{-1}(\beta)$ from vector v or $-v$, depending on whether v is a descent direction or not, respectively. Thus, we have the following two cases:

1. If v is a descent direction, we prune the positive spanning direction set D to obtain the pruned direction set D_k^p as

$$D_k^p = \left\{ d \in D : d^T v \geq \beta \|d\| \|v\| \right\}. \tag{4.3.1}$$

2. If v is not a descent direction, the pruned direction set D_k^p is obtained as

$$D_k^p = \left\{ d \in D : d^T v \leq -\beta \|d\| \|v\| \right\}. \tag{4.3.2}$$

Since we do not want to evaluate the (computationally expensive) gradient of f, we judge whether or not v is a descent direction by using a sufficiently small step size $\alpha > 0$. That is, if $f(x_k + \alpha v) < f(x_k)$, we consider v a descent direction. Otherwise, we do not consider v a descent direction. It is noteworthy that Abramson et al. [2] use the gradient to prune the positive spanning direction set D by means of (4.3.1) with $v = -\nabla f(x_k)$ and $\beta = 0$. The use of gradients, however, may not be appropriate in the case where they are computationally so expensive that a derivative-free method such as a PS method becomes a method of choice.

Algorithm 4.3.1 below describes the steps of the HPS method. In the ADD step, we may use either the *Random* method or the *Orthogonal* method described in the previous section. In practice, we prefer to use the *Random* method since the *Orthogonal* method is computationally more expensive. Moreover, the positive spanning direction set D used in the PS step can be set either $\{e_1, \ldots, e_n, -e_1, \ldots, -e_n\}$ or $\{e_1, \ldots, e_n, -e\}$, where $e_i \in R^n$ is the ith unit vector in R^n and $e \in R^n$ is the vector of ones.

Algorithm 4.3.1. *HPS($f, x_0, \Delta_0, \alpha, \sigma$)*

1. ***Initialization.*** *Choose an initial solution x_0, fix an initial mesh size $\Delta_0 > 0$, choose the shrinkage coefficient σ of the mesh size from $(0, 1)$, fix a sufficiently small step size $\alpha > 0$, set the pruning control parameter $\beta \in (-1, 1)$, and set the iteration counter $k := 0$.*

2. ***ADD.*** *Calculate the vector v at x_k as in (4.2.1). If $f(x_k + \Delta_k v) < f(x_k)$, then set $x_{k+1} := x_k + \Delta_k v$, and go to Step 5.*

3. ***PS.*** *If $f(x_k + \alpha v) < f(x_k)$, then use (4.3.1) to obtain D_k^p. Otherwise, use (4.3.2) to obtain D_k^p. Evaluate f on the trial points $\{p_j = x_k + \Delta_k d_j : d_j \in D_k^p, j = 1, \ldots, |D_k^p|\}$.*

4. ***Parameter Update.*** *If $\min_{1 \le j \le |D_k^p|} f(p_j) < f(x_k)$, then set $x_{k+1} := \arg\min_{1 \le j \le |D_k^p|} f(p_j)$. Otherwise, decrease Δ_k through the rule $\Delta_{k+1} := \sigma \Delta_k$.*

5. *If the stopping condition is satisfied, then terminate. Otherwise, let $k := k+1$ and return to step 2.*

To implement Algorithm 4.3.1, we have to determine a proper value of the pruning control parameter β. We use the standard $2n$ directions, $D = \{e_1, \ldots, e_n, -e_1, \ldots, -e_n, \}$ as a positive spanning direction set. In this case, a proper value for β can be chosen from $(-1, \frac{1}{\sqrt{n}})$ to guarantee that the pruned direction set D_k^p contains at least one direction. In the following, we study the tuning of parameter β through some numerical experiments. Four values $\beta = \frac{1}{\sqrt{n}}, \frac{1}{2\sqrt{n}}, 0, \frac{-1}{2\sqrt{n}}$ have been chosen to make some numerical simulations using Rosenbrock function R_2, De Joung function DJ, and Zakharov functions $Z_n, n = 2, 4, 10, 20$, see Appendix A for the analytical formulae of these test functions. Note that the global minimum values of all these functions are 0. Figures 4.2–4.7 show that $\beta = \frac{1}{\sqrt{n}}$ generally gives faster convergence toward the global minima than the other values of β. It is notable from these figures that the performance of the HPS method for the function R_2 is different from that for other functions. Figure 4.3 shows that the HPS method with $\beta = \frac{1}{\sqrt{n}}$ works well in the early stage of the search, while it suffers from slow convergence in the later stage compared with the method using other values. However, this difference in performance is expected since the HPS method uses the ADD method and descent-type methods usually suffer from slow convergence when applied to R_2.

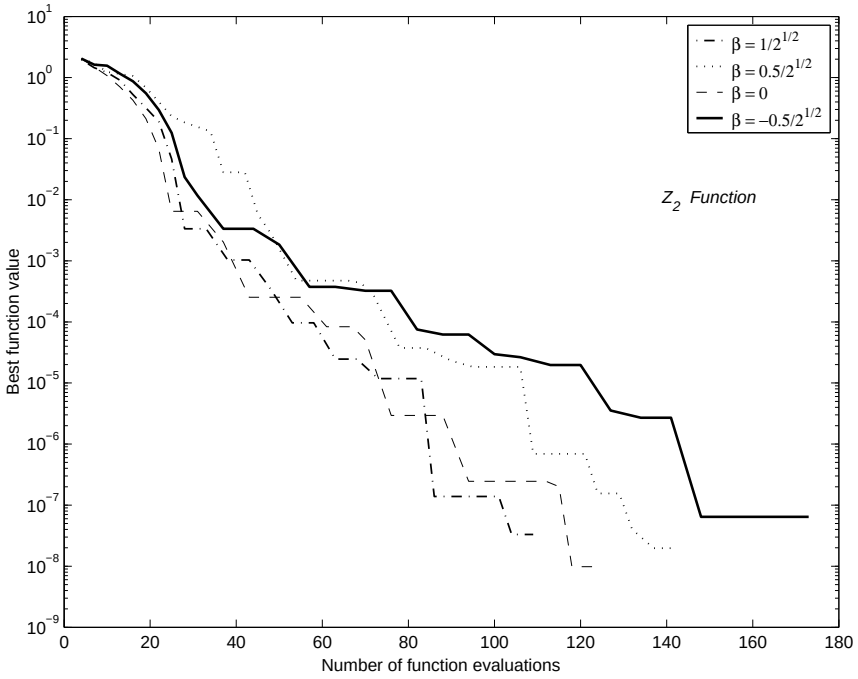

Figure 4.2: The HPS performance for Zakharov function Z_2.

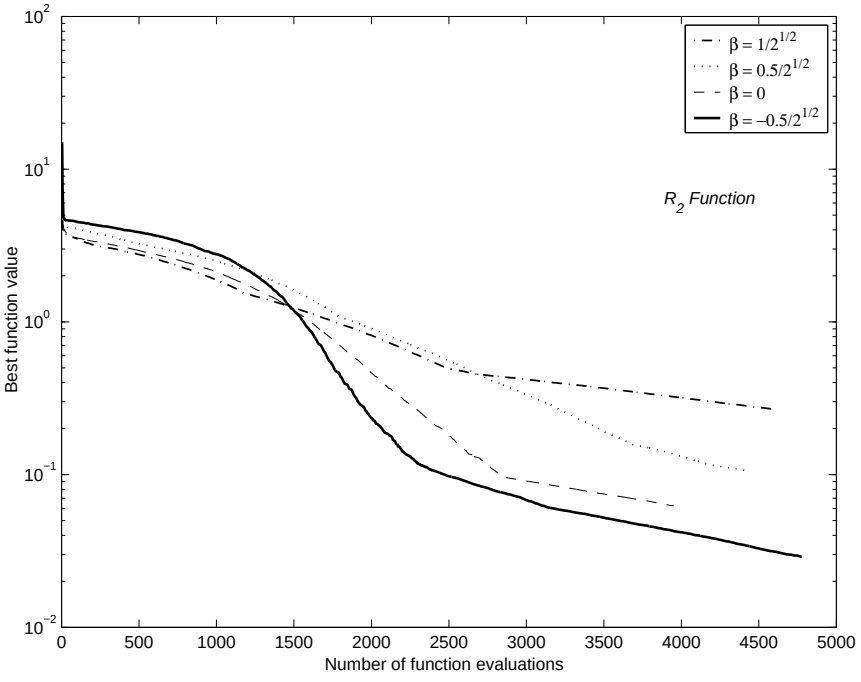

Figure 4.3: The HPS performance for Rosenbrock function R_2.

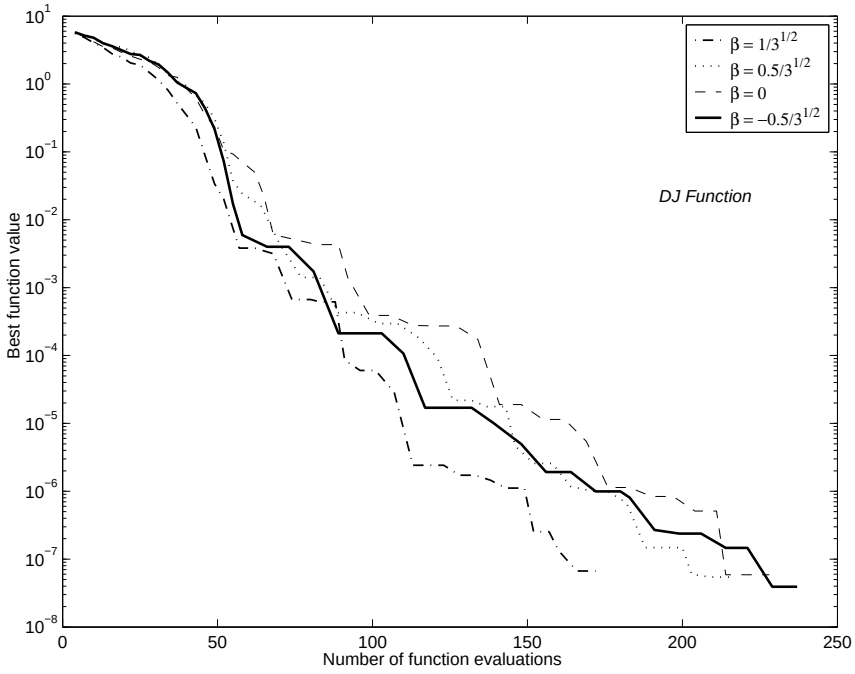

Figure 4.4: The HPS performance for De Joung function.

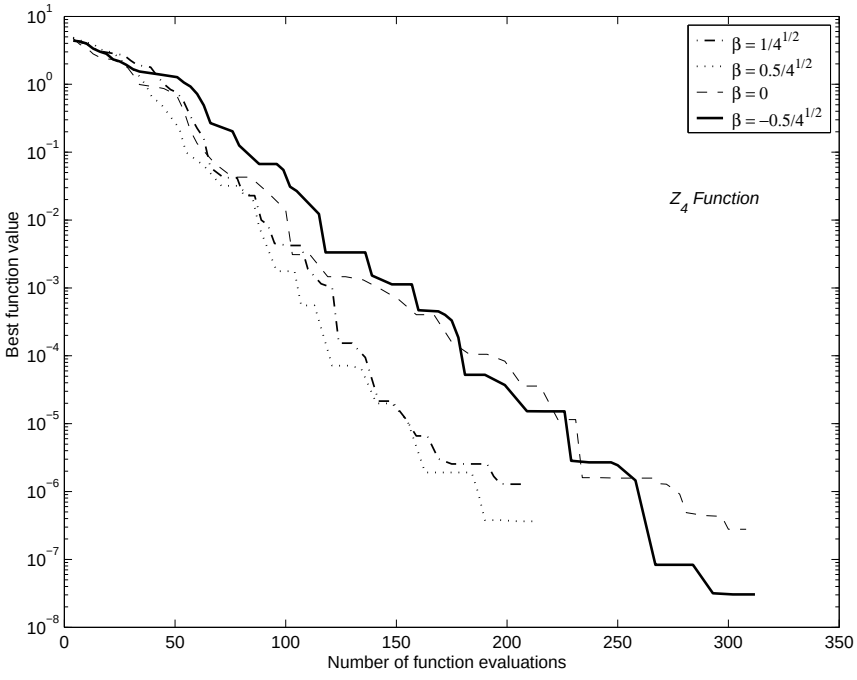

Figure 4.5: The HPS performance for Zakharov function Z_4.

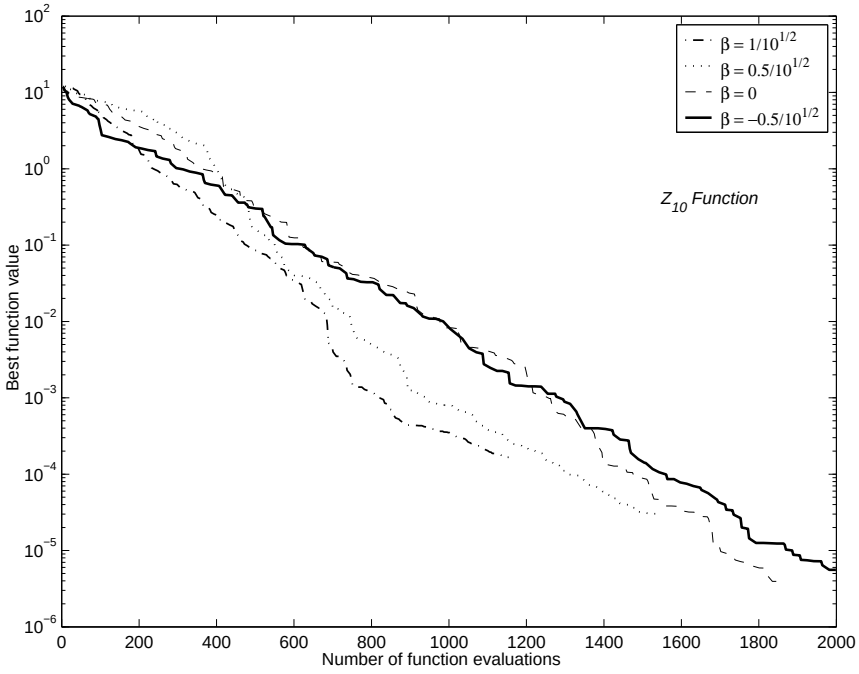

Figure 4.6: The HPS performance for Zakharov function Z_{10}.

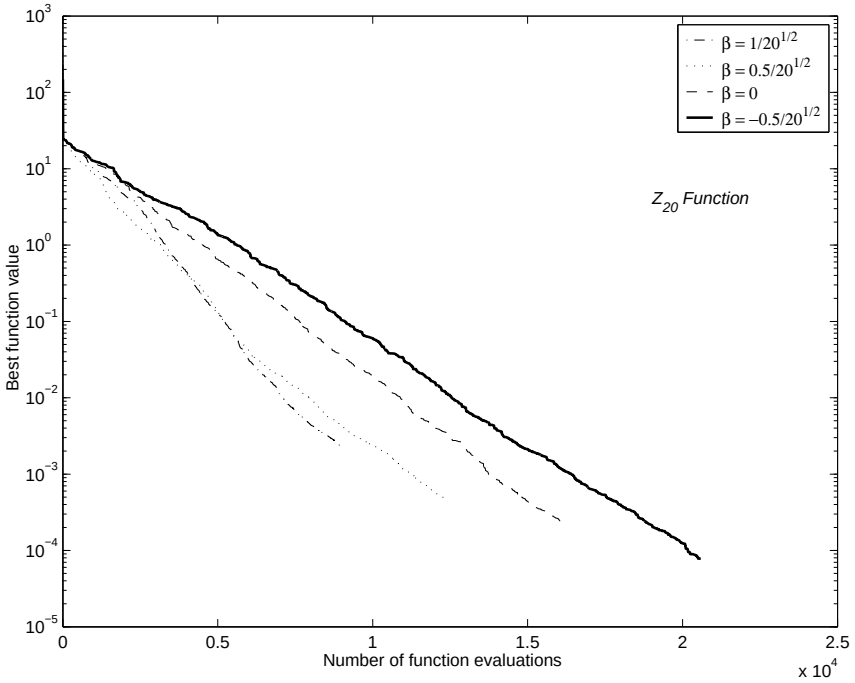

Figure 4.7: The HPS performance for Zakharov function Z_{20}.

Table 4.2: Results of PS and HPS for Zakharov functions

	No. f-evals.		Best f value	
f	PS	HPS	PS	HPS
Z_2	97	132	3.0E–9	2.6E–8
Z_4	353	234	9.1E–9	3.1E–7
Z_{10}	7821	1547	2.6E–6	8.4E–5
Z_{20}	50000+	11140	4.4E–2	1.7E–3

A question that arises when we test the performance of the HPS method is to what extent the ADD used in HPS helps the PS to get better results. To examine this issue, we compare the HPS method with the plain PS method on Zakharov functions $Z_n, n = 2, 4, 10, 20$. We use the standard $2n$ directions, $D = \{e_1, \ldots, e_n, -e_1, \ldots, -e_n\}$, to generate the pattern search directions in each method. Table 4.2 shows the best function value (Best f value) and the number of function evaluations (No. f-evals.) achieved by each method. We use the same starting points for all methods. Since there is no random step in the PS method, it was run only once for each problem. On the other hand, the HPS method was run 100 times and the Best f value and the No. f-evals. are the average of these 100 trials. The pruning control parameter β was set equal to $\frac{1}{\sqrt{n}}$. Moreover, the shrinkage coefficient σ is set equal to 0.5, which is the standard value of the shrinkage coefficient in direct search methods. The initial mesh size should be chosen big enough for more efficient local search, so that we set Δ_0 equal to 1. The step size α is set equal to 10^{-3}, which is small enough to avoid misleading the search especially in the vicinity of a local minimum. The iteration was terminated in Step 5 when the mesh size became smaller than 10^{-4}, or the number of function evaluations exceeded $50,000$.

From the results shown in Table 4.2, we may observe that using the ADD in the HPS method can reduce the number of function evaluations in the plain PS method especially for higher dimensional problems.

4.4 Simulated Annealing HPS

In this section we give the details of our main hybrid method SAHPS. The SA approach is combined with the HPS to form the hybrid method SAHPS, which is expected to have a higher ability to detect global minima. At each major iteration of the SAHPS method,

we first repeat the simulated annealing acceptance trials m_1 times. In each time, a trial point is generated by using an exploring point to guide the SA search along a promising direction and to avoid making a blind random search. Specifically, we generate an exploring point z_k close to the current iterate x_k and a SA trial is generated along the direction $sign(f(x_k) - f(z_k))(z_k - x_k)$, with a certain step size. If more than m_{ac} out of m_1 trials are accepted, then we immediately proceed to the next major iteration of SAHPS. Otherwise, within the same major iteration, we repeat the HPS iterations m_2 times. In the early stage of the search, the diversification is more needed than the intensification, however, the converse is needed in the final stage of the search. Since the HPS represents the intensification part of the SAHPS, it is better to initialize the value of m_2 at a moderate value and increase it while the search is going on. In the end of the search, we complete the algorithm by applying a fast local search method to refine the best point obtained by the search so far. We prefer to use the Kelley's modification [51, 52] of the Nelder-Mead method [72] in this final step.

More detailed and formal description of the SAHPS method is shown in the following Algorithm 4.4.1. The setting of parameters used in this algorithm will be discussed later in next section.

Algorithm 4.4.1. *SAHPS($f, x_0, \Delta_0, r, \alpha, \epsilon$)*

1. ***Initialization.*** *Choose an initial solution x_0, fix an initial mesh size $\Delta_0 > 0$, choose the shrinkage coefficient σ of the mesh size from $(0, 1)$, fix the SA trial point radius r, fix a sufficiently small step size $\alpha > 0$, and fix a sufficiently small neighborhood radius $\epsilon > 0$. Fix the cooling schedule parameters; initial temperature $T_{\max}$, epoch length M, cooling reduction ratio $\lambda \in (0.5, 0.99)$, and minimum temperature $T_{\min}$. Set the temperature $T := T_{\max}$.*

2. ***The main iteration.*** *Repeat the following Global SA Search (Step 2.1) m_1 times. If more than m_{ac} out of m_1 trial points are accepted, then skip the Local HPS (Step 2.2) and proceed to Step 3.*

2.1 Global SA search. *Given the current iterate x_k, generate an exploring point z_k randomly in the neighborhood of x_k with radius ϵ. Generate a trial*

point x_{SA} in the neighborhood of the current solution x_k by

$$x_{SA} = \begin{cases} x_k + \eta r\,(z_k - x_k)\,/\,\|z_k - x_k\|\,, & \text{if } f\,(z_k) \le f\,(x_k)\,, \\ x_k - \eta r\,(z_k - x_k)\,/\,\|z_k - x_k\|\,, & \text{otherwise,} \end{cases}$$

where η is a random number in $(0.1, 1)$. Evaluate f on the trial point x_{SA}, and accept it, i.e. $x_{k+1} := x_{SA}$, if

i. $\Delta f := f(x_{SA}) - f(x_k) < 0$, *or*

ii. $\Delta f \ge 0$, *and* $p = \exp\left(\frac{-\Delta f}{T}\right) \ge u$, *where u is a random number in $(0, 1)$.*

2.2. Local HPS. *Repeat the following procedure m_2 times.*

2.2.a. ADD. *Calculate the vector v at x_k as in (4.2.1). If $f\,(x_k + \Delta_k v) < f\,(x_k)$, then set $x_{k+1} := x_k + \Delta_k v$, and proceed to the next iteration of the Local HPS loop.*

2.2.b. PS. *If $f\,(x_k + \alpha v) < f\,(x_k)$, then use (4.3.1) to obtain D_k^p. Otherwise, use (4.3.2) to obtain D_k^p. Evaluate f on the trial points $\{p_j := x_k + \Delta_k d_j : d_j \in D_k^p,\ j = 1, \ldots, |D_k^p|\}$.*

2.2.c. Parameter update. *If $\min_{1 \le j \le |D_k^p|} f(p_j) < f\,(x_k)$, then set $x_{k+1} := \arg\min_{1 \le j \le |D_k^p|} f(p_j)$. Otherwise, decrease Δ_k through the following rule:*

$$\Delta_{k+1} := \sigma \Delta_k. \tag{4.4.1}$$

3. *If the epoch length, which corresponds to M iterations of Global SA Search, is not achieved, then go to Step 2.*

4. *If the cooling schedule is completed ($T \le T_{\min}$) or the function values of two consecutive improvement trials become close to each other or the number of iterations exceeds $50n$, then go to Step 5. Otherwise, decrease the temperature*

by setting $T := \lambda T$, increase m_2 slightly, decrease r slightly, and go to Step 2.

5. *From the best point found, apply the modified Nelder-Mead method [51, 52].*

4.5 Experimental Results

4.5.1 Setting of Parameters

Below we elaborate on the implementation of Algorithm 4.4.1.

Initial trial. The initial point x_0 is chosen randomly from the predetermined range $[L, U]$ of the initial points for each test function, where

$$[L, U] = \{x \in R^n : l_i \leq x_i \leq u_i,\ i = 1, \ldots, n\}.$$

Cooling schedule. Generally, choosing a proper cooling schedule is not a trivial task. Our cooling schedule is designed based on some common choice of parameters suggested in the literature, or according to our observation gained through some preliminary numerical experiments. First, the initial temperature $T_{\max}$ is set large enough to make the initial probability of accepting transition close to 1. Beside the initial point x_0, another point $\widetilde{x}_0$ is generated in a neighborhood of x_0 to calculate $T_{\max}$ as

$$T_{\max} := -\frac{1}{\ln(0.9)} \left| f(\widetilde{x}_0) - f(x_0) \right|.$$

The cooling ratio λ is normally chosen to be between 0.9 and 0.99 [57]. Actually, in the original SA method, Kirkpatrick et al. [53] suggested $\lambda = 0.95$, which has become a common choice. However, in our experiments, we observed that the results obtained with $\lambda = 0.95$ were not significantly different from those with $\lambda = 0.9$. The main reason for this insignificant difference is due to the use of refining local search method at the final stage. Since setting λ equal to 0.95 is more computationally costly, we set λ equal to 0.9. A common choice of the number of trials allowed at each temperature, which is called epoch length M, is to let it depend on the size of the problem [58]. Although Kirkpatrick et al. [53] set the value of M equal to n, our preliminary experiments have revealed that setting M equal to $2n$ fits the SAHPS algorithm well. Therefore, we set M equal to $2n$. In the implementation of SA, the cooling schedule is terminated when the temperature reaches a fixed minimum value

$T_{\min}$ [58]. We observed that setting $T_{\min}$ equal to $\min(10^{-3}, 10^{-3}T_{\max})$ can give a complete cooling schedule in the sense that the acceptance probability at the end is almost zero.

Neighborhood radius. The neighborhood radius ϵ, which is used in generating the exploring points z_k in the *Global SA Search* and in generating the exploring points used to compute vector v in the ADD step, is set equal to 10^{-3}.

SA trial point radius. The radius r, which is used in generating the SA trial points, is initialized as $r_0 := \max_{1\le i\le n}(u_i - l_i)/5$ to fit the search domain of each test function, and then r is reduced in parallel to the reduction of temperature T by setting $r := \max\{0.95r, 0.02r_0\}$.

HPS parameters. The mesh size is initialized as $\Delta_0 := \max_{1\le i\le n}(u_i - l_i)/10$, and when no improvement is achieved, its shrinkage factor σ in (4.4.1) is set equal to 0.7. Actually, the standard value of the shrinkage coefficient in direct search methods is 0.5 but, in our experiments, we observed that using the value 0.7 gives more ability of efficient exploration than the value 0.5, because the latter may decrease the step size prematurely before reaching a proper exploration process. We set the step size α used in Step 2.2.a equal to 10^{-3}. The pattern search strategy described in Section 4.3 is adopted in the *Local HPS* step.

Loop repetitions. The repetition numbers of the *Global SA Search* and *Local HPS* steps, m_1 and m_2, are both set equal to n initially, which is equal to the half of the epoch length M. However, m_2 is updated as $m_2 := \min\{5n, 1.05m_2\}$ in Step 4 at every major iteration. The control parameter m_{ac}, the desired number of accepted points in the *Global SA Search* step, is set equal to 1.

Termination conditions. Beside the completeness of the cooling schedule, Algorithm 4.4.1 may be terminated in Step 4, if the difference between the function values of two consecutive improvement trials becomes less than $Tol = 10^{-8}$, or the number of iterations exceeds $It_{max} = 50n$.

In Table 4.3, we summarize all parameters used in the SAHPS algorithm with their assigned values.

4.5.2 Numerical Results

Algorithm 4.4.1 was programmed in Matlab and applied to 19 well-known test functions [15, 37], see Appendix A. For each function, this Matlab code was run 100 times with different initial points. To judge the success of a trial, we used the condition

Table 4.3: SAHPS Parameters

Parameter	Definition	Value
$T_{\max}$	maximum (initial) temperature	$-\frac{1}{\ln(0.9)}\left\|f(\widetilde{x}_0)-f(x_0)\right\|$
$T_{\min}$	minimum temperature	$\min\left(10^{-3}, 10^{-3}T_{\max}\right)$
λ	cooling ratio	0.9
M	epoch length	$2n$
ϵ	neighborhood radius	10^{-3}
r_0	initial radius for generating SA trial points	$\min_{1\le i\le n}\left(u_i-l_i\right)/5$
Δ_0	initial mesh size	$\min_{1\le i\le n}\left(u_i-l_i\right)/10$
σ	reduction factor of mesh size	0.7
α	step size for checking descent directions	10^{-3}
m_1	*Global SA Search* repetition number	n
m_2	*Local HPS* repetition number	$\begin{cases}\text{initial: } n\\ \text{update: } \min\{5n, 1.05m_2\}\end{cases}$
m_{ac}	number of accepted points in *Global SA Search*	1
It_{max}	maximum number of iterations	$50n$
Tol	termination tolerance	10^{-8}

$$\left|f^*-\widehat{f}\right| < \epsilon_1\left|f^*\right|+\epsilon_2, \tag{4.5.1}$$

where $\widehat{f}$ refers to the best function value obtained by SAHPS, f^* refers to the known exact global minimum value, and ϵ_1 and ϵ_2 are small positive numbers. We set ϵ_1 and ϵ_2 equal to 10^{-4} and 10^{-6}, respectively. The average number of function evaluations (Av. f-evals.) and the average errors (Av. Error) reported in Table 4.4 are those for the successful trials. It is noteworthy that, for some of the functions that fail to achieve the 100% success rate, the success rate can be improved by relaxing the maximum number of iterations or by slowing down the cooling schedule. For example, the success rate for *SH* function can be improved to 95% with Av. f-evals. of 822 and Av. Error of 9E–6, if we set the maximum number of iterations equal to $100n$ instead of $50n$.

To complete the testing of the SAHPS method, we compare it with other SA-based methods, Enhanced Simulated Annealing (ESA) [85] and Direct Search Simulated Annealing (DSSA) proposed in Chapter 2. The results of ESA are taken from its original reference [85], as well as [15]. The results of DSSA are the same as those shown in Table 2.2.

The results shown in Table 4.4 indicate that SAHPS generally outperforms the ESA. Since ESA is a plain SA method without any combination with a local search method, we may conclude that hybridizing HPS with SA significantly improves the performance of SA.

Table 4.4: Results of SAHPS and other SA methods

	Av. f-evals.			Av. Error		
f	SAHPS	ESA	DSSA	SAHPS	ESA	DSSA
RC	318	–	118	4E–7	–	4E–7
ES	432(96%)	–	1442(93%)	5E–9	–	3E–9
GP	311	783	261	5E–9	9E–3	4E–9
B_1	346	–	252	8E–9	–	5E–9
HM	278	–	225	5E–8	–	5E–8
SH	450(86%)	–	457(94%)	9E–6	–	9E–6
Z_2	276	15820	186	7E–9	–	4E–9
R_2	357	796	306	6E–9	–	4E-9
DJ	398	–	273	6E–9	–	5E–9
$H_{3,4}$	517(95%)	698	572	2E–6	5E–4	2E–6
$S_{4,5}$	1073(48%)	1137(54%)	993(81%)	3E–7	4E–3	2E–6
$S_{4,7}$	1059(57%)	1223(54%)	932(84%)	4E–5	8E–3	6E–7
$S_{4,10}$	1031(48%)	1189(50%)	992(77%)	1E–5	4E–2	1E–5
Z_5	716	96799	914	8E–9	–	5E–9
R_5	1104(91%)	5364	2685	7E–9	–	3E-9
$H_{6,4}$	997(72%)	1638	1737(92%)	2E–6	6E–2	2E–6
GR	795	–	1830(90%)	8E–9	–	5E–9
Z_{10}	2284	15820	12501	3E–8	2E–3	7E–9
R_{10}	4603(87%)	12403	16785	2E-8	4E-2	7E-9

On the other hand, the comparison between SAHPS and DSSA does not seem to yield a definitive answer. The performance of DSSA is better for the problems with $n < 5$ but SAHPS outperforms DSSA in higher dimensional problems, i.e., $n \geq 5$, in terms of the number of function evaluations.

4.6 Conclusion

In this chapter, we have presented a new hybrid global search method in which a direct search method is combined with the SA procedure to remedy the slow convergence of the latter method. Two new methods have been introduced to design the SAHPS method; one is the ADD method that produces an approximate descent direction, the other is the HPS method that is used to make a local exploratory search in the main SAHPS method. The

latter has turned out to be particularly effective because the HPS method shows a superior performance in reducing the computational expense of the plain PS method.

Chapter 5

Directed TS for Unconstrained Global Optimization

5.1 Introduction

Tabu Search (TS) is one of the recent metaheuristics originally developed for combinatorial optimization problems [31, 33]. TS has shown an appropriate performance when applied to these problems [31]. However, contributions of TS to solving continuous optimization problems are still very limited compared with other metaheuristics like Simulated Annealing and Genetic Algorithms. In this chapter, we introduce a TS approach that deals with the continuous unconstrained optimization problem

$$\min_{x \epsilon R^n} \ f(x), \tag{5.1.1}$$

where f is a generally nonconvex, real valued function defined on R^n. Specifically, we present continuous versions of TS called Directed Tabu Search (DTS) by hybridizing TS with direct search methods. The role of direct search methods is to stabilize the search especially in the vicinity of a local minimum. Specifically, instead of using completely blind random search in generating neighborhood trial moves, appropriate direct search strategies are responsible to generate these neighborhood moves. Moreover, new implementations of TS elements are employed in the proposed method.

Since the main presentation of Glover [28, 29], a lot of studies have emerged in the area of TS. The majority of these studies are related to combinatorial optimization problems

and relatively few attempts have been made to deal with continuous optimization problems [3, 10, 15, 20, 21, 26, 50]. One of the earliest TS methods was presented by Hu [50] for constrained optimization problems. Cvijovic and Klinowski [20, 21] extented and generalized the TS to functions with variables that may be continuous or, if discrete, need not take integer values. Battiti and Tecchiolli [10] introduced an interesting continuous TS method called the Continuous Reactive Tabu Search. Their method tries to locate the most promising boxes, and then starting points for the local search are generated within those boxes. Al-Sultan and Al-Fawzan [3] gave a hybrid method that combines TS with the local search method of Hooke and Jeeves.

Recently, intensive studies on continuous TS have been conducted in [15, 26]. In [15], Chelouah and Siarry introduced a new algorithm called Enhanced Continuous Tabu Search (ECTS), which aims to follow Glover's basic approach as closely as possible. In order to cover a wide domain of possible solutions, the ECTS algorithm first performs a diversification search to locate the most promising areas. When the most promising areas are located, the algorithm proceeds to an intensification search within one promising area of the solution space. In [26], Franze and Speciale presented a novel TS algorithm that explores a grid of points with a distance dynamically adjusted. When it collapses to a local minimum, it continues the local search from that point while accepting some non-improving points to allow the exploration of new regions in the search space.

The DTS method proposed in this chapter differs from the previous continuous TS methods in many aspects. In the DTS method, three search procedures are employed; Exploration, Diversification and Intensification. In the Exploration Search, a new local search procedure is introduced to generate trial moves, based on the well-known Nelder-Mead method [72] and the heuristic pattern search method proposed in Chapter 4. Moreover, novel concepts of TS memory elements called Tabu Regions (TRs), Semi-TRs and a multi-ranked Tabu List (TL) are introduced to provide anti-cycling rules. Another memory element called Visited Regions List (VRL) is also introduced as a tool for the Diversification Search to diversify the search to unvisited areas of the solution space. Finally, assuming that one of the best points obtained by the Exploration and Diversification Searches is close to a global minimum, the Intensification Search is applied again at the final stage to refine the elite solutions visited so far. Actually, the proposed Diversification and Intensification Searches try to follow some known strategies from the high level TS with a long term memory. Moreover, the DTS can be classified as a multi-start method. The multi-start methods aim to construct powerful search procedures by guidance of global exploration and local searches; as surveys for multi-start methods the reader is referred to [62, 63, 84]. Multi-start methods have been

successfully applied to both nonlinear global optimization and combinatorial problems, see [63] and references therein. Finally, the numerical results reported below show the promise of the proposed method especially in producing high quality solutions.

The rest of this chapter is organized as follows. The next section gives a detailed description of the proposed TS memory elements. In Section 5.3, we introduce neighborhood and local search strategies used to generate the trial moves. The main DTS method is presented elaborately in Section 5.4. Section 5.5 discusses the implementation of the proposed method and reports comprehensive experimental results. The conclusion of the contribution of this chapter makes up Section 5.6.

5.2 TS Memory Elements

The concept of memory plays a major role in TS, especially when it is used in a kind of learning process as in high level TS with long term memory. Using an effective memory conception in intensification and diversification schemes makes TS behave as an intelligent search technique [31]. The optimization search methods can be classified in two categories; point-to-point methods and population-based methods. TS belongs to the first category. Keeping the diversity is one of the main problems that face the point-to-point methods compared with the population-based methods. However, the long term memory in TS makes it competitive with the population-based methods in keeping the diversity. In TS with long term memory, the search can be restarted from new diverse solutions whenever the diversification is needed, or can be intensified to improve the elite solutions whenever the intensification is needed. These TS concepts of diversification and intensification have turned out to be effective in many combinatorial optimization problems, see [31, 61] for example. In this section, we introduce some new conceptions and implementations of the TS memory elements to continuous optimization problems. First, we let the multi-ranked Tabu List (TL) be a set of some visited solutions. The points in the TL are ranked and saved according to their recency and their objective function values. Therefore, some positions in the TL are kept for the best visited solutions, which helps an intensification scheme to refine the search from these best solutions at the final stage. Around each solution saved in the TL, two types of regions are specified in the search space. The first one is a Tabu Region (TR) in which no new trial point is allowed to be generated. The other is a Semi-Tabu Region (Semi-TR) that comprises a surrounding region around TR. The main role of the Semi-TRs is to generate neighboring trial points in a special way so that returning back to a visited TR is avoided

when the trial solution lies inside a Semi-*TR*. Another memory element introduced in this section is the Visited Region List (*VRL*). The centers of the visited regions and the frequency of visiting these regions are saved in the *VRL* in order to direct a diversification scheme to explore the space outside these visited regions.

5.2.1 Multi-Ranked Tabu List (*TL*)

Some of the previously visited solutions are stored in the *TL*. Let $TL = \{t_i\}_{i=1}^{L}$. The elements in *TL* are ranked in ascending order according to their recency using the rank indices $I_i^r,\ i = 1, \ldots, L$, i.e., if the most recent element in *TL* is t_k, then $I_k^r = 1$, while if the most ancient element is $t_{k'}$, then $I_{k'}^r = L$. Also, the elements in *TL* are ranked in ascending order according to their objective function values using another set of rank indices $I_i^{fv},\ i = 1, \ldots, L$, i.e., if the best element in *TL* is t_j, then $I_j^{fv} = 1$, and if the worst element is $t_{j'}$, then $I_{j'}^{fv} = L$. In the ordering, ties are broken arbitrarily. We consider the *TL* to be a fuzzy set and associate its elements $\{t_i\}_{i=1}^{L}$ the membership values:

$$m_i = \max\left\{m_i^r, m_i^{fv}\right\},\ i = 1, \ldots, L, \tag{5.2.1}$$

where $m_i^r, m_i^{fv} \in [0, 1]$ are the recency and the function-value ranked values, respectively, for element t_i and they are computed as follows:

- *The recency ranked value* m^r. We use a linear ranking procedure that gives the most recent element the maximum ranked value $\eta_{\max}$ and the most ancient element the minimum ranked value $\eta_{\min}$, where $0 \leq \eta_{\min} < \eta_{\max} \leq 1$. Specifically, the recency ranked value for each element of *TL* is given by

$$m_i^r = \eta_{\min} + (\eta_{\max} - \eta_{\min})\left(\frac{L - I_i^r}{L - 1}\right),\ i = 1, \ldots, L.$$

- *The function-value ranked value* m^{fv}. To avoid reserving excessively many positions in the *TL* for the best elements and to give the recency some priority, this procedure ranks only $\overline{L}$ best elements so that the best element is given the ranked value $\mu_{\max}$, and the worst $L - \overline{L} + 1$ elements are given the ranked value $\mu_{\min}$, where $1 \leq \overline{L} \leq L$ and $0 \leq \mu_{\min} < \mu_{\max} \leq 1$. Specifically, the function-value ranked value for each element of *TL* is given by

$$m_i^{fv} = \begin{cases} \mu_{\min} + (\mu_{\max} - \mu_{\min})\left(\frac{\overline{L} - I_i^{fv}}{\overline{L} - 1}\right), & \text{if } I_i^{fv} = 1, \ldots, \overline{L}, \\ \mu_{\min}, & \text{if } I_i^{fv} = \overline{L} + 1, \ldots, L. \end{cases}$$

The Tabu Regions (*TRs*) are defined to be spheres with radius r_{TR} and their centers being the points of *TL,* where $r_{TR} > 0$. For each *TR*, we define Semi-*TR* to be the surrounding region around this *TR* with outer radius r_{STR} from its center, where $r_{STR} > r_{TR}$. If a trial solution lies in Semi-*TRs*, then a specific procedure is applied to create special neighborhood trial points to avoid returning back to a vicinity of a previously visited solution. We suggest the following procedure for this purpose.

Procedure 5.2.1. *Trial Solution Generation in Semi-TRs*

1. *Let a trial point x lie in ν Semi-TRs with centers $t_1, \ldots, t_\nu$. Compute the centroid $\overline{t}$ of the Semi-TRs' centers and the maximum distance $d_{\max}$ between x and these centers, i.e.,*

$$\begin{aligned} \overline{t} &= \frac{1}{\nu} \sum_{i=1}^{\nu} t_i, \\ d_{\max} &= \max_{i=1,\ldots,\nu} \{\|x - t_i\|\}. \end{aligned}$$

2. *Construct neighborhood search directions that are parallel to the coordinate axes but point towards the direction $x - \overline{t}$, i.e., the neighborhood search directions are determined as sign $\left((x)_i - \left(\overline{t}\right)_i\right) e_i$, $i = 1, \ldots, n$, where $e_i \in R^n$ is the ith unit vector in R^n. Neighborhood trial points are generated along these search directions with a suitable step size $\beta > 0$. In the case of $\nu > 1$, the step size β should be chosen greater than $d_{\max} + r_{TR}$ in order to avoid generating trial points inside a TR.*

Figure 5.1 illustrates how Procedure 5.2.1 works when a solution x lies in Semi-*TRs* in two dimensions. In this example, the solution x lies in two Semi-*TRs* with center t_1 and t_2. According to Procedure 5.2.1, the neighborhood search directions d_1 and d_2 are constructed to follow the vector $\left(x - \overline{t}\right)$, where $\overline{t}$ is the centroid of the Semi-*TRs*' centers. It is noteworthy that the step size used to generate a trial point along search directions d_1 and d_2 is chosen to be greater than $r_{TR} + \max\{\|x - t_1\|, \|x - t_2\|\}$, to make sure that the close *TRs* with centers t_1 and t_2 will not be hit.

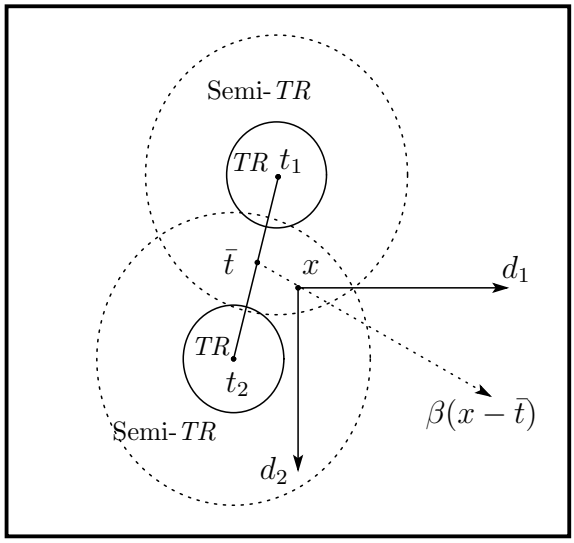

Figure 5.1: Neighborhood search from a point in Semi-*TRs*.

5.2.2 Visited Region List (*VRL*)

Some historical information about the previously visited regions is stored in the *VRL*. More specifically, the center ζ_i of a visited region, which is a sphere with radius ρ_i, and the frequency φ_i of visiting this region comprise the information stored in the *VRL*, i.e., $VRL = \{(\zeta_i, \rho_i, \varphi_i)\}_{i=1}^{M}$, where M is the number of all listed visited regions. The information in *VRL* will be used to direct the search towards new regions whenever the current TS procedure fails to get improvement or whenever a diversification scheme is needed. As a diversification scheme, we try to generate new trial points outside the visited regions. However, generating trial points near to more frequently visited regions is discouraged. To this end, a function $\Phi(\varphi)$ is introduced to distinguish between more and less frequently visited regions. Specifically, we define the function Φ as

$$\Phi(\varphi) = \gamma \left(1 - e^{-\gamma(\varphi-1)}\right), \tag{5.2.2}$$

where $\gamma \in (0, 1]$ is a given constant. Note that the function Φ is strictly increasing and bounded above by the value γ. We will describe the role of γ in the diversification scheme after we state Procedure 5.2.2 below.

In the following, we suggest a procedure that uses the *VRL* information to generate a new solution. The procedure allows accepting a trial point outside the visited regions, especially the more frequently visited ones.

Procedure 5.2.2. *Diverse Solution Generation*

1. *Generate a trial point x randomly in the search domain of f.*

2. *Compute the quantities $d_i = \|x - \zeta_i\|/(1 + \Phi(\varphi_i))$, $i = 1, \ldots, M$, where $\Phi(\varphi)$ is defined by (5.2.2). If $\min_{1 \le i \le M} d_i/\rho_i \ge 1$, then accept x. Otherwise, return to Step 1.*

A point x is accepted by Procedure 5.2.2 if it satisfies $\|x - \zeta_i\|/\rho_i \ge 1 + \Phi(\varphi_i)$ for all $i = 1, \ldots, M$. This means that no point can be accepted inside a previously visited region. Moreover, a point close to more frequently visited regions is hardly accepted. Therefore, the higher the value of γ is, the lower the possibility of accepting a point close to more frequently visited regions is. To avoid infinitely cycling in Procedure 5.2.2, we may also terminate it after a predetermined number of iterations and return with x corresponding to the maximum of the values of $\min_{1 \le i \le M} d_i/\rho_i$ over all iterations.

5.3 Neighborhood-Local Search Strategies

To explore the region around a solution and to generate the next move, we use neighborhood and local search strategies in which direct search methods are employed. Specifically, two search strategies are introduced to handle that job; Nelder-Mead Search (NMS) strategy and Adaptive Pattern Search (APS) strategy, which are based on the well-known Nelder-Mead method [72] and the heuristic pattern search method proposed in Chapter 4, respectively. These neighborhood-local search strategies are invoked to generate trial points in the Exploration Search of the DTS method. More specifically, two types of trial points are generated by the neighborhood-local search strategy; neighborhood trial points and local trial points, which are needed in the Neighborhood Search and Local Search Steps, respectively, in Algorithms 5.4.1 and 5.4.2 stated in the next section. First, p trial points $\{y_i\}_{i=1}^{p}$ are generated in a neighborhood of the current solution x. This procedure is called a neighborhood search, and the trial points $\{y_i\}_{i=1}^{p}$ are called neighborhood trial points. Then, we try to improve the neighborhood trial points $\{y_i\}_{i=1}^{p}$ by executing another search procedure, which is called a local search, to generate q trial points $\{y_{p+i}\}_{i=1}^{q}$, which are called local trial points. The details of the neighborhood-local search strategies, NMS and APS, are given below.

5.3.1 Nelder-Mead Search (NMS) Strategy

In the NMS strategy, we generate $p(= n)$ neighborhood trial points $\{y_i\}_{i=1}^{n}$, and $q(= 1$ or $0)$ local trial point. The neighborhood trial points are generated along search directions parallel to the coordinates axes starting from the current solution x with suitable step sizes. If the current solution x lies in a Semi-TR or in Semi-TRs, we apply Procedure 5.2.1 to construct the search directions and the step sizes. Otherwise, we construct search directions parallel to the coordinate axes in a random way, i.e., each of them is parallel to a positive or a negative coordinate direction. To generate a local trial point, we construct a simplex S that consists of the current solution x and the current n neighborhood trial points $\{y_i\}_{i=1}^{n}$, i.e., $S = \{x, y_1, \ldots, y_n\}$. Some iterations of the NM method are applied starting from S. If an improvement point is obtained from these NM iterations, then we set the local trial point y_{n+1} equal to this improvement point, i.e., $q = 1$. Otherwise, there is no trial point, i.e., $q = 0$.

For more explanation of the NMS strategy, we show an example in two dimensions in Fig. 5.2. Given the current solution x, two neighborhood trial points y_1 and y_2 are generated in a neighborhood of x as in Fig. 5.2 (a). To find a local trial point, we construct a simplex whose vertices are $S = \{x, y_1, y_2\}$, as in Fig. 5.2 (b). Assuming that the worst point in S is y_2, we apply the Nelder-Mead method operations described in Fig. 5.2 (c) to find a better movement. If one exists, we refer to this better movement as a local trial point.

5.3.2 Adaptive Pattern Search (APS) Strategy

The main idea behind the APS strategy is based on the approximate descent direction (ADD) method proposed in Chapter 4. We implement a similar procedure as in ADD method to produce a new adaptive direction from standard pattern directions. Specifically, we construct n pattern directions parallel to the coordinate axes emanating from the point x and generate n trial points $\{y_i\}_{i=1}^{n}$ along these directions with a suitable step size. The adaptive direction v, along which we may expect to decrease the function value, is computed using these trial points as follows:

$$v = \sum_{i=1}^{n} \omega_i u_i, \tag{5.3.1}$$

where

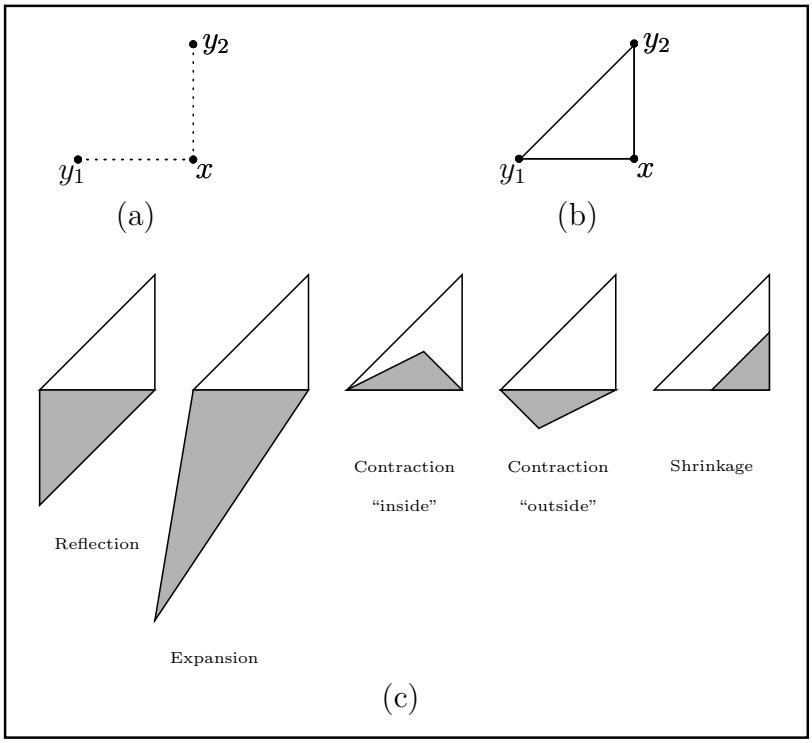

Figure 5.2: NMS strategy in two dimensions.

$$\begin{aligned} \omega_i &= \frac{\Delta f_i}{\sum_{j=1}^{n} |\Delta f_j|}, \quad i = 1, 2, \ldots, n, \\ u_i &= -\frac{(y_i - x)}{\|y_i - x\|}, \quad i = 1, 2, \ldots, n, \\ \Delta f_i &= f(y_i) - f(x), \quad i = 1, 2, \ldots, n. \end{aligned}$$

In the APS strategy, we generate $p(= n)$ neighborhood trial points $\{y_i\}_{i=1}^{n}$ using the standard pattern directions, and $q(= 2)$ local trial points using an adaptive pattern direction. More specifically, we construct n pattern directions parallel to the coordinate axes emanating from the current solution x and generate n neighborhood trial points $\{y_i\}_{i=1}^{n}$ along these directions with some step size. The adaptive pattern direction v at x is computed using (5.3.1). Two local trial points y_{n+1} and y_{n+2} are generated along the vector v with two different step sizes.

An example in two dimensions is illustrated in Fig. 5.3 to describe the APS strategy. Two neighborhood trial points y_1 and y_2 are generated in a neighborhood of the current solution x as in Fig. 5.3 (a). An approximate descent direction v is computed at x using the points y_1 and y_2, as in (5.3.1). If we assume that x is better than y_1 and y_2, then, by means of (5.3.1), the vector v is composed toward the vectors $x - y_1$ and $x - y_2$ with weights inversely proportional to $|f(x) - f(y_1)|$ and $|f(x) - f(y_2)|$, see Fig. 5.3 (b). Finally, two local trial points y_3 and y_4 are generated along the vector v with two different step sizes in order to explore the area along the promising direction v as in Fig. 5.3 (c).

5.4 Directed Tabu Search (DTS)

In this section, we describe some details about how a TS method is modified with the memory elements and neighborhood-local search strategies introduced in Sections 5.2 and 5.3 to compose the DTS method.

In the DTS method, three basic search procedure are used; Exploration, Diversification and Intensification search procedures. In the Exploration Search, we use the neighborhood-local search strategies, which are described in Section 5.3, to explore the solution space. Moreover, the multi-ranked *TL*, *TR* and Semi-*TR* restriction rules are applied to avoid revisiting recently visited solutions or being entrapped in local minima. Then, the Diversification Search is needed in order to diversify the search to other areas of the solution space that may have been overlooked in the Exploration Search. We use the *VRL* and Procedure

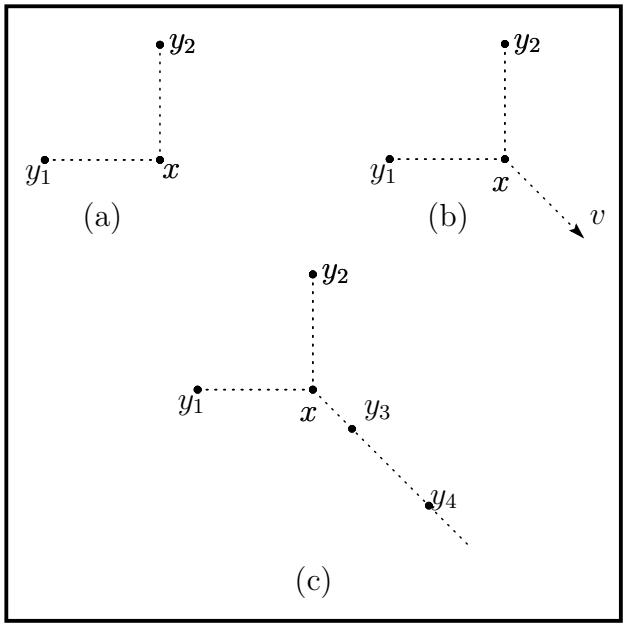

Figure 5.3: APS strategy in two dimensions.

5.2.2 to manage the Diversification Search. Finally, in order to explore the close regions around the best points visited so far, the Intensification Search is applied to refine these best points. These search procedures are applied in such a way that they give the DTS method a better chance to explore the search space efficiently. Actually, the Exploration and Diversification search procedures are assembled to compose the DTS main loop and are repeated until the termination conditions are satisfied. Moreover, the Exploration Search procedure is included as an inner loop within the diversification loop. We will use the superscript $j = 0, 1, \ldots,$ to represent the main loop iteration counter, the subscript $k = 0, 1, \ldots,$ to represent the inner loop iteration counter, and $x_k^{(j)}$ to denote a general iterate. In other words, the Exploration and Diversification search procedures compose a multi-start procedure with a long term memory. At the final stage, the Intensification Search procedure based on elite TS is needed to complete the DTS method. The main structure of the DTS method is shown in Fig. 5.4. More detailed description of the search procedures is given below.

5.4.1 Exploration-Diversification Loop

The main loop of the DTS method, which consists of Exploration and Diversification Searches, starts with an initial solution $x_0^{(0)}$. In each main loop iteration j, the Exploration Search pro-

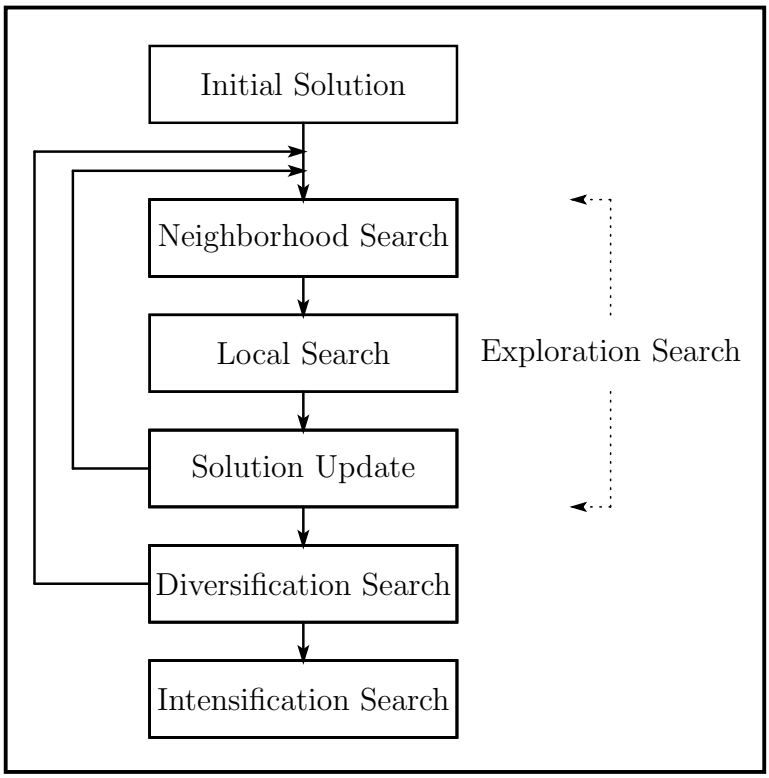

Figure 5.4: Main structure of the DTS method.

cedure is repeatedly applied to obtain improvement by means of neighborhood-local search strategies, and then the Diversification Search procedure is applied to locate a new initial trial point $x_0^{(j+1)}$, from which the Exploration Search is restarted again. This main loop is repeated at most ℓ_{main} times, where ℓ_{main} is a predetermined positive integer.

Exploration Search. The Exploration Search starts with an initial solution $x_0^{(j)}$ at each main loop iteration j. In each iteration of the Exploration Search, a neighborhood-local search (NMS or APS) strategy is used to generate n neighborhood trial points $\{y_i\}_{i=1}^n$ in a neighborhood of the current iterate $x_k^{(j)}$. If a better movement is found among these trial points, we update the current iterate and proceed to the next inner loop iteration. Otherwise, i.e., $x_k^{(j)}$ is still better than all neighborhood trial points, the neighborhood-local search strategy continues to generate q local trial points $\{y_{n+i}\}_{i=1}^q$, where $q = 0$ or 1 in the NMS strategy and $q = 2$ in the APS strategy. Then, the current iterate $x_k^{(j)}$ is updated to be the best of neighborhood and local trial points, i.e., $x_{k+1}^{(j)} := \arg\min_{i=1,\ldots,n+q} \{f(y_i)\}$. *TL* is also updated by letting $x_k^{(j)}$ replace the element with the smallest membership value. If a new region is reached, then *VRL* should be updated by adding the information on this region. This Exploration Search loop is repeated at most ℓ_{inner} times, where ℓ_{inner} is a predetermined positive integer.

Diversification Search. The Diversification Search is carried out when the Exploration Search either spends the inner iterations ℓ_{inner} times or fails to obtain an improvement in some consecutive iterations. With the current *VRL*, Procedure 5.2.2 is applied to generate a trial point $x_0^{(j+1)}$ in some new region. Then the Exploration Search is repeated again starting from $x_0^{(j+1)}$.

5.4.2 Intensification Search

According to the principle of the multi-ranked *TL,* it reserves the best points visited so far. In order to improve these points, we complete the DTS method by applying another local search method starting from some of these points, which we call Intensification Search. We use Kelley's modification [51, 52] of the Nelder-Mead (NM) method as a local search method for this task.

5.4.3 Main Algorithm

We have two versions of the DTS method; DTS_{NMS} and DTS_{APS} that use the NMS strategy and APS strategy, respectively, as neighborhood-local search strategies. First, a specific and

formal description of DTS_{NMS} is given in the following Algorithm 5.4.1.

Algorithm 5.4.1. *$DTS_{NMS}(f, x_0^{(0)})$*

1. ***Initialization.*** *Choose positive integers ℓ_{main}, ℓ'_{main}, ℓ_{inner} and ℓ'_{inner}. Choose an initial solution $x_0^{(0)}$, and set TL and VRL to be empty.*

2. ***Exploration-Diversification Search (Main Loop).*** *Let $j := 0$ and repeat this main loop until ℓ'_{main} consecutive main iterations fail to obtain improvement or the main loop iteration counter j exceeds ℓ_{main}.*

2.1. Exploration Search(NMS) (Inner Loop). *Let $k := 0$ and repeat this inner loop until ℓ'_{inner} consecutive inner iterations fail to obtain improvement or the inner loop iteration counter k exceeds ℓ_{inner}.*

2.1.1. Search Directions. *If the current iterate $x_k^{(j)}$ lies in Semi-TRs, use Procedure 5.2.1 to construct search directions $\{d_i\}_{i=1}^n$ and to choose step sizes $\{\Delta_i\}_{i=1}^n$. Otherwise, construct search directions $d_i := (-1)^{\tau_i} e_i$, $i = 1, \ldots, n$, where $e_i \in R^n$ is the ith unit vector in R^n and τ_i is a random integer number, and choose suitable step sizes $\{\Delta_i\}_{i=1}^n$.*

2.1.2. Neighborhood Search. *Generate n neighborhood trial points $y_i := x_k^{(j)} + \Delta_i d_i$, $i = 1, \ldots, n$. Whenever a better movement is found during this process, stop generating points, set $x_{k+1}^{(j)}$ equal to this better movement, and go to Step 2.1.4.*

2.1.3. Local Search. *Apply n iterations of the NM method starting from the simplex $S := \left\{x_k^{(j)}, y_1, \ldots, y_n\right\}$. If an improvement point is obtained from these NM iterations, set local trial point y_{n+1} equal to this improvement point, and set $q := 1$. Otherwise, set $q := 0$. Set $x_{k+1}^{(j)} := \arg\min_{i=1,\ldots,n+q} \{f(y_i)\}$.*

2.1.4. Parameter Update. *Let $x_k^{(j)}$ replace the element with the smallest*

membership value in TL and re-rank the TL elements using (5.2.1). Update the VRT and set $k := k + 1$.

2.2. Diversification Search. *Generate a trial point* $x_0^{(j+1)}$ *using Procedure 5.2.2. Update the TL and VRT, and set* $j := j + 1$.

3. Intensification Search. *Apply the Kelley's modification [51] of the NM method starting from some elite solutions in the TL.*

The DTS_{APS} algorithm is the same as Algorithm 5.4.1 except Step 2.1, which should be changed to follow the structure of APS strategy as shown in Algorithm 5.4.2. Only Step 2.1 of Algorithm 5.4.2 is stated since other steps are the same as the corresponding steps in Algorithm 5.4.1.

Algorithm 5.4.2. $DTS_{APS}(f, x_0^{(0)})$

...

2.1. Exploration Search(APS) *(Inner Loop).* *Let* $k := 0$, *initialize a vector* v *to be a random vector in* R^n, *and repeat this inner loop until* ℓ'_{inner} *consecutive inner iterations fail to obtain improvement or the inner loop iteration counter* k *exceeds* ℓ_{inner}.

2.1.1. Search Directions. *If the current iterate* $x_k^{(j)}$ *lies in Semi-TRs, use Procedure 5.2.1 to construct search directions* $\{d_i\}_{i=1}^n$ *and to choose step sizes* $\{\Delta_i\}_{i=1}^n$. *Otherwise, construct search directions* $d_i := sign(v_i)e_i$, $i = 1, \ldots, n$, *where* $e_i \in R^n$ *is the ith unit vector in* R^n *and* v_i *is the ith component of* v, *and choose suitable step sizes* $\{\Delta_i\}_{i=1}^n$.

2.1.2. Neighborhood Search. *Generate* n *neighborhood trial points* $y_i := x_k^{(j)} + \Delta_i d_i$, $i = 1, \ldots, n$. *Whenever a better movement is found during this process, stop generating points, set* $x_{k+1}^{(j)}$ *equal to this better movement, and go to Step 2.1.4.*

2.1.3. Local Search. *Compute the direction v at $x_k^{(j)}$ using $\{y_i\}_{i=1}^n$ as in (5.3.1). Choose two suitable step sizes α_1 and α_2 to generate local trial points $y_{n+i} = x_k^{(j)} + \alpha_i v / \|v\|,\ i = 1, 2$. Set $x_{k+1}^{(j)} := \arg\min_{i=1,\ldots,n+2} \{f(y_i)\}$.*

2.1.4. Parameter Update. *Let $x_k^{(j)}$ replace the element with the smallest membership value in TL and re-rank the TL elements using (5.2.1). Update the VRT and set $k := k + 1$.*

...

5.5 Implementation and Experiments

In this section, we give more details about the implementation as well as the experimental results of the DTS algorithms.

5.5.1 Setting Parameters

In this subsection, we discuss the suggested values of the parameters needed in the implementation of the DTS algorithms and the sensitivity of these parameters. First, the initial trial solution $x_0^{(0)}$ is chosen randomly from the predetermined range $[\mathfrak{L}, \mathfrak{U}]$ of the initial points for each test function, where $[\mathfrak{L}, \mathfrak{U}] := \{x \in R^n : l_i \le x_i \le u_i,\ i = 1, \ldots, n\}$. The other parameters can be classified into the following groups.

- ***TR* and Semi-*TR* parameters:** the radius r_{TR} of each *TR*, and the outer radius r_{STR} of the Semi-*TR*.
- ***TL* parameters:** the number L of elements in *TL*, the maximum and minimum recency ranked values $\eta_{\max}$ and $\eta_{\min}$, respectively, the number $\overline{L}$ of the function-value ranked elements, and the maximum and minimum function-value ranked values $\mu_{\max}$ and $\mu_{\min}$, respectively.
- ***VRL* parameters:** the radii $\rho_j,\ j = 1, \ldots, M$, of the visited regions.
- **Step sizes:** the step sizes $\Delta_i,\ i = 1, \ldots, n$, used in generating neighborhood trial points in DTS$_{NMS}$ and DTS$_{APS}$, and α_1 and α_2 used to generate local trial points in DTS$_{APS}$.

- **Diversification trials:** the parameter γ used in (5.2.2) and the maximum number It_{max} of iterations allowed in Procedure 5.2.2.

- **Intensification trials:** the number N_{best} of best points that are used in the Intensification Search.

- **Termination conditions:** the loop termination numbers ℓ_{main}, ℓ'_{main}, ℓ_{inner} and ℓ'_{inner}.

Proper values of these parameters have been studied through extensive numerical experiments by using the functions Branin (RC), Goldstein&Price (GP), Rosenbrock (R_2) and Zakhrov (Z_2) and (Z_5). In the tuning parameters experiments, we have tried to find a standard setting of the DTS parameters which is problem-independent as much as possible. Moreover, some relations between the parameters have been discussed to guide the user to set the DTS parameters whenever new implementations of the DTS algorithm are invoked. Below, we highlight the conclusion we got from the tuning parameters experiments.

First, the values of the parameter γ that we have studied are $0.10, 0.15, 0.2, \ldots, 0.4$. Recall that the main role of this parameter is to avoid generating a new diverse trial solution near to the more frequently visited regions. Since large γ may lead to a big area surrounding the more frequently visited regions left without exploration, we did not test a value of γ higher than 0.4. The performance of the DTS algorithms is almost the same for all runs using the above-mentioned values of γ. Moreover, at the end of running the DTS algorithms on many test functions, the centers of the visited regions listed in the VRL are distributed almost uniformly for all tested values of γ. However, the value $\gamma = 0.25$ produces slightly more scattered distributions than the other values. Therefore, we set γ equal to 0.25. The parameter It_{max} is set equal to $100n$.

Most of the DTS parameters listed above are distance parameters. These distance parameters are r_{TR}, r_{STR}, ρ, $\Delta_i, i = 1, \ldots, n$, α_1 and α_2. Note that the radii $\rho_j,\ j = 1, \ldots, M$, of all visited regions are set equal to ρ. For more accurate setting of the values of these distance parameters, we consider the following:

1. Since Semi-TRs are surrounding TRs, we let $r_{STR} > r_{TR}$. For easily escaping from TRs and Semi-TRs, it is desirable to let $\Delta_i > r_{STR}$. Moreover, to avoid producing too many small visited regions, we let $\rho > \Delta_i$. This means that the desirable order of the distance parameters is $r_{TR} < r_{STR} < \Delta_i < \rho$. Since the step sizes α_1 and α_2, are used to search the area along an approximate descent direction, it is appropriate to let one of them be smaller than the usual step size Δ_i and the other be greater than Δ_i.

Table 5.1: Distance parameters.

Parameter	Tested values	Suggested value
r_{TR}	$0.01\delta,\ 0.02\delta,\ 0.03\delta,\ 0.04\delta$	0.01δ
Δ_i	$0.08\delta,\ 0.1\delta,\ 0.12\delta$	0.1δ
ρ	$0.15\delta,\ 0.2\delta,\ 0.25\delta$	0.15δ

2. To keep the distance parameters in the above order, we let their values relate to only one parameter δ which is the diameter of the range $[\mathfrak{L}, \mathfrak{U}]$ defined as $\delta := \max_{1\leq i\leq n}(u_i - l_i)$.

The performance of the DTS algorithms were tested using different values for these parameters through many test functions. The suggested values of these parameters are given in Table 5.1, and r_{STR} is set equal to $2r_{TR}$. The performance of the DTS algorithms were almost insensitive with regard to all tested values of the distance parameters. In Table 5.1, we also suggest the value for each parameter which produces the best performance. For more efficient search, the step sizes may be randomly chosen close to some fixed mean values, rather than being set at fixed values. Specifically, we set the step sizes as follows:

$$\begin{aligned}\Delta_i &= (0.1 + 0.025\omega_i)\delta,\ i = 1, \ldots, n,\\ \alpha_1 &= (0.1 - 0.05\theta_1)\delta,\\ \alpha_2 &= (0.1 + 0.05\theta_2)\delta,\end{aligned}$$

where $\omega_i,\ i = 1, \ldots, n$, are random numbers from the interval $(-1, 1)$, and θ_1 and θ_2 are random numbers from the interval $(0, 1)$.

For the TL parameters, the values $5n$, $6n$, $7n$ and $8n$ and the values $2n$, $3n$ and $4n$ have been tested as possible choices for L and $\overline{L}$, respectively. The performance of the DTS algorithms using these values of L and $\overline{L}$ is almost the same. So, to avoid storing unnecessary information, we set L and $\overline{L}$ equal to the least possible values, i.e., $L = 5n$ and $\overline{L} = 2n$. The other TL parameters are set as $\eta_{\max} = \mu_{\max} = 1$ and $\eta_{\min} = \mu_{\min} = 1/L$.

The parameter N_{best} is set equal to 1 because the numerical results show that the best point found in the Exploration-Diversification Search is close to global minima for most of the test function.

The last group of parameters are related to the termination conditions. Actually, choosing sufficient large values for the loop termination numbers ℓ_{main}, ℓ'_{main}, ℓ_{inner} and ℓ'_{inner} is highly needed to avoid premature termination of the method. The numerical results have shown

that the lowest values of these parameters that can give an acceptable performance of the DTS algorithms are $\ell_{main} = \ell_{inner} = 5n$, and $\ell'_{main} = \ell'_{inner} = 2n$. However, higher values for these numbers can increase the ability of finding global minima for some difficult test problems.

5.5.2 Numerical Results

In this subsection, we discuss the performance of the DTS algorithms through two main experiments. The first experiment is to compare the results obtained by the DTS_{NMS} and DTS_{APS} and then compare the best version of them with other continuous versions of TS. In the second experiment, the performance of the best DTS algorithms is also compared with other metaheuristics.

Numerical Results of DTS and other TS methods

To examine the performance of the DTS algorithms DTS_{NMS} and DTS_{APS} , we tested them on some well known functions [25, 39] listed as Set A in Table 5.2 , see Appendix A for more details of these test functions. The characteristics of these test functions are diverse enough to cover many kinds of difficulties that arise in unconstrained global optimization problems. To complete the evaluation of the DTS algorithms, they should be compared with other continuous versions of TS. However, it is not easy to show complete and fair comparisons due to the lack of some information especially on the quality of solutions obtained by those continuous TS methods. Therefore, we try to compare our algorithms with other continuous TS methods in terms of the ability of obtaining global minima, the cost of function evaluations and the quality of computed solutions. Three continuous TS methods chosen to compare with the DTS algorithms are continuous reactive TS (CRTS) [10], Enhanced Continuous Tabu Search (ECTS) [15], and TS-based algorithm called DOPE [26]. The ECTS and DOPE methods are the most recent continuous TS methods and the quality of computed solutions are stated clearly in their original references.

For each function in Set A, we applied the DTS codes 100 times with different starting points. For all these test functions, we used the same values of the DTS parameters as those presented in Subsection 5.5.1. Moreover, we used the same condition as that used by ECTS [15] to judge the success of a trial, which is given by

Table 5.2: Test functions (Set A).

No.	f	Function name	n	No.	f	Function name	n
1	RC	Branin RCOS	2	9	$S_{4,5}$	Shekel	4
2	ES	Easom	2	10	$S_{4,7}$	Shekel	4
3	GP	Goldstein&Price	2	11	$S_{10,7}$	Shekel	4
4	SH	Shubert	2	12	Z_5	Zakharov	5
5	Z_2	Zakharov	2	13	R_5	Rosenbrock	5
6	R_2	Rosenbrock	2	14	$H_{6,4}$	Hartmann	6
7	DJ	De Joung	3	15	Z_{10}	Zakharov	10
8	$H_{3,4}$	Hartmann	3	16	R_{10}	Rosenbrock	10

$$\left| f^* - \widetilde{f} \right| < \epsilon_1 \left| f^* \right| + \epsilon_2, \tag{5.5.1}$$

where $\widetilde{f}$ refers to the best function value obtained by the algorithm, f^* refers to the exact global minimum, and ϵ_1 and ϵ_2 are set equal to 10^{-4} and 10^{-6}, respectively. Note that the conditions for successful trials are not stated for CRTS and DOPE in the original references [10, 26].

The results of the two versions of DTS method, DTS_{NMS} and DTS_{APS}, are reported in Table 5.3. These results represent the average number of function evaluations (Av. f-evals.) with minimum and maximum numbers in parentheses, the average errors (Av. Error) and the success rates (Suc.) for each function. The average number of function evaluations and the average error only relate to successful trials. The results shown in Table 5.3 reveal that the performance of DTS_{APS} is consistently better than DTS_{NMS} in terms of function evaluations and the ability of obtaining global minima. Moreover, it seems that DTS_{NMS} suffers from the curse of dimensionality as is seen from the Av. f-evals. for higher dimensional problems.

Table 5.4 compares DTS_{APS} with the above-mentioned continuous TS methods in terms of the average number of function evaluations. The results of CRTS, ECTS and DOPE methods are taken from their original references [10, 15, 26]. The percentages in parentheses represent the success rates of reaching global minima. The quality of the computed solutions by those methods except the CRTS method is shown in Table 5.5, where the errors are measured in terms of function values at the computed and exact solutions. The quality of the produced solutions by the CRTS method is not stated clearly in [10], but it is only said that the statistical error on the CRTS is about 3%. Before judging the comparison of these

Table 5.3: Results of DTS algorithms.

	DTS_{NMS}			DTS_{APS}		
f	*Av. f-evals.(min/max)*	*Av. Er.*	*Suc.*	*Av. f evals.(min/max)*	*Av. Er.*	*Suc.*
RC	274(252/296)	4e–7	100%	212(181/243)	4e–7	100%
ES	271(202/285)	5e–9	30%	223(156/244)	4e–9	82%
GP	293(276/324)	5e–9	88%	230(207/282)	5e–9	100%
SH	298(282/319)	9e–6	44%	274(260/307)	9e–6	92%
Z_2	273(247/291)	6e–9	100%	201(183/225)	5e–9	100%
R_2	358(272/489)	6e–9	100%	254(207/321)	5e–9	100%
DJ	650(600/694)	5e–9	100%	446(393/516)	4e–9	100%
$H_{3,4}$	670(613/789)	2e–6	97%	438(389/493)	2e–6	100%
$S_{4,5}$	1426(1342/1473)	7e–7	39%	819(669/989)	3e–7	75%
$S_{4,7}$	1425(1372/1487)	4e–5	29%	812(675/973)	4e–5	65%
$S_{4,10}$	1438(1340/1493)	1e–5	22%	828(706/963)	1e–5	52%
Z_5	2458(2301/2597)	6e–9	100%	1003(903/1093)	7e–9	100%
R_5	2895(2523/3473)	7e–9	75%	1684(1326/2093)	6e–9	85%
$H_{6,4}$	3978(3618/4308)	2e–6	68%	1787(1489/2036)	2e–6	83%
Z_{10}	16392(14235/17821)	2e–8	100%	4032(3689/4809)	2e–8	100%
R_{10}	19139(16844/22416)	2e–8	78%	9037(6701/12879)	2e–8	85%

methods, some remarks are made in regard to the reported success rates of ECTS and the termination condition of DOPE.

- The ECTS method uses condition (5.5.1) to test the success of a trial [15]. However, the results marked by (⊛) in Tables 5.4 and 5.5 seem to contain some inconsistencies. In fact, from condition (5.5.1), the average errors for functions R_2, R_5 and Z_5 must be less than 10^{-6} because $f^* = 0$ for all these functions. However, in Table 5.5, they are reported to be greater than 10^{-6}. For instance, the average error for function R_5 in Table 5.5 is 0.08, which means that there are some trials that did not satisfy condition (5.5.1). Nevertheless, the rate of success is reported to be 100%. Moreover, the results of ECTS for functions $RC, ES, GP, H_{3,4}$ and $H_{6,4}$ in Tables 5.4 and 5.5 also contain similar inconsistencies.

- According to [26], DOPE is terminated when either a maximum number of function evaluations is reached or the global minimum (if it is a priori known) is found. Since the information on global minima is not available in practice, we did not use the latter termination condition in our numerical experiments. This termination condition may explain the extremely small number of function evaluations of DOPE for some test functions.

From these remarks, the comparisons of the DTS methods with the ECTS and DOPE methods do not seem to yield a definitive fair answer. However, in terms of the quality of computed solutions, the DTS$_{APS}$ algorithm seems to outperform ECTS and DOPE as shown in Table 5.5. Moreover, the DTS$_{APS}$ algorithm seems to outperform ECTS in terms of the number of function evaluations for functions in Set A. However, the drawback we have noticed on the DTS algorithms is its deterioration in high dimensional problems ($n > 30$). Actually, this can be expected since the search in the DTS algorithms is mainly controlled by direct search methods and it has been shown, for instance, in [54] that the latter methods deteriorate with the increase of the dimension, i.e., suffer from the curse of dimensionality. To show the limit of deterioration of the DTS performance with the dimensionality, we report some results for high dimensional problems. The results have been obtained by running the Matlab code of DTS$_{APS}$, with the parameter setting given in Subsection 5.5.1, on Pentium 2.8-GHz machine. For Rosenbrock R_{50} function, the DTS$_{APS}$ algorithm converged to a point close to the global minimum with function value 4.46×10^{-7} using $510,505$ function evaluations in 1085 CPU seconds. For Zakharov Z_{50} and Rosenbrock R_{100} functions, the DTS$_{APS}$ algorithm obtained points not so close to the global minimum at distances 1.404 and 4.1057 with function values 1.972 and 4.106 using $177,125$ and $3,202,879$ function evaluations in $1,043$ and $15,270$ CPU seconds, respectively. These results of Z_{50} and R_{100} are the best among 5 runs. For Zakharov Z_{100}, the DTS$_{APS}$ algorithm failed to obtain a point near the global minimum by 5 runs using the same setting of parameters. As far as the results in Table 5.4 common to all methods are concerned, CRTS may be considered the best among the continuous TS methods in terms of the ability of obtaining global minima and the number of function evaluations.

The performance of DTS$_{APS}$ against other metaheuristics

The performance of DTS$_{APS}$ is compared with other metaheuristics using the test functions listed as Set B in Table 5.6 [59], see Appendix A for more details of these test functions. We choose two other metaheuristics proposed for the continuous optimization problem; Genetic algorithm for numerical optimization of constrained problems (Genocop III) [67], and Scatter Search (SS) [59].

Table 5.4: Average numbers of function evaluations for continuous TS methods.

	DTS_{APS}	ECTS	DOPE	CRTS	
f				CRTS_{Ave}	CRTS_{Min}
RC	212	$245^{\circledast}$	31	38	41
ES	223(82%)	$1284^{\circledast}$	290	–	–
GP	230	$231^{\circledast}$	248	248	171
SH	274(92%)	370	466	–	–
Z_2	201	195	81	–	–
R_2	254	$480^{\circledast}$	692	–	–
DJ	446	338	131	–	–
$H_{3,4}$	438	$548^{\circledast}$	135	513	609
$S_{4,5}$	819(75%)	825(75%)	–	812	664
$S_{4,7}$	812(65%)	910(80%)	–	960	871
$S_{4,10}$	828(52%)	898(75%)	–	921	693
Z_5	1003	$2254^{\circledast}$	424	–	–
R_5	1684(85%)	$2142^{\circledast}$	2512	–	–
$H_{6,4}$	1787(83%)	$1520^{\circledast}$	421	750	1245
Z_{10}	4032	4630	8695	–	–
R_{10}	9037(85%)	15720(85%)	5133	–	–

Table 5.5: Average errors for continuous TS methods.

f	DTS_{APS}	ECTS	DOPE	f	DTS_{APS}	ECTS	DOPE
RC	4e–7	$5e–2^{\circledast}$	5e–2	$S_{4,5}$	3e–7	1e–2	–
ES	4e–9	$1e–2^{\circledast}$	1e–2	$S_{4,7}$	4e–5	1e–2	–
GP	5e–9	$2e–3^{\circledast}$	2e–3	$S_{4,10}$	1e–5	1e–2	–
SH	9e–6	1e–3	1e–3	Z_5	7e–9	$4e–6^{\circledast}$	4e–6
Z_2	5e–9	2e–7	2e–7	R_5	6e–9	$8e–2^{\circledast}$	8e–2
R_2	5e–9	$2e–2^{\circledast}$	2e–2	$H_{6,4}$	2e–6	$5e–2^{\circledast}$	5e–2
DJ	5e–9	3e–8	3e–8	Z_{10}	2e–8	2e–7	2e–2
$H_{3,4}$	2e–6	$9e–2^{\circledast}$	9e–2	R_{10}	2e–8	2e–2	2e–7

Table 5.6: Test functions (Set B).

No.	f	Function name	n	No.	f	Function name	n
1	RC	Branin RCOS	2	21	$PS_{8,18,44,114}$	Power Sum	4
2	B_2	Bohachevsky	2	22	$H_{6,4}$	Hartmann	6
3	ES	Easom	2	23	SC_6	Schwefel	6
4	GP	Goldstein&Price	2	24	T_6	Trid	6
5	SH	Shubert	2	25	T_{10}	Trid	10
6	BL	Beale	2	26	RT_{10}	Rastrigin	10
7	BO	Booth	2	27	G_{10}	Griewank	10
8	MT	Matyas	2	28	SS_{10}	Sum Squares	10
9	HM	Hump	2	29	R_{10}	Rosenbrock	10
10	SC_2	Schwefel	2	30	Z_{10}	Zakharov	10
11	R_2	Rosenbrock	2	31	RT_{20}	Rastrigin	20
12	Z_2	Zakharov	2	32	G_{20}	Griewank	20
13	DJ	De Joung	3	33	SS_{20}	Sum Squares	20
14	$H_{3,4}$	Hartmann	3	34	R_{20}	Rosenbrock	20
15	CV	Colville	4	35	Z_{20}	Zakharov	20
16	$S_{4,5}$	Shekel	4	36	PW_{24}	Powell	24
17	$S_{4,7}$	Shekel	4	37	DP_{25}	Dixon&Price	25
18	$S_{4,10}$	Shekel	4	38	L_{30}	Levy	30
19	$P_{4,0.5}$	Perm	4	39	SR_{30}	Sphere	30
20	$P^0_{4,0.5}$	Perm	4	40	AK_{30}	Ackley	30

Table 5.7: Average optimality gap values.

f-evals.	100	500	1000	5000	10000	20000	50000
Genocop III[1]	5.37E+25	2.39E+17	1.13E+14	636.37	399.52	320.84	313.34
Genocop III[2]	1335.45	611.30	379.03	335.81	328.66	324.72	321.20
Scatter Search	134.45	26.34	14.66	4.96	3.60	3.52	3.46
DTS_{APS}	5.04E+04	43.06	24.26	4.22	1.80	1.70	1.29

[1] Average values for all test functions.
[2] Average values for all test functions except function 23.

We define the optimality gap (GAP) [59] as the quantity on the left-hand side of (5.5.1). Table 5.7 shows the average GAP for all 40 test functions in Set B. In Table 5.7, the figures related to Genocop III and SS are taken from [59] and represent the average GAP for all test functions in Set B at intermediate stages during the search. Since the DTS_{APS} algorithm consists of two complementary parts (Exploration-Diversification Search and Intensification Search), its results in Table 5.7 are the average GAP for all test functions in Set B obtained by running the DTS_{APS} code 7 times for each test function with the termination condition that the number of function evaluations exceeds 100, 500, 1000, 5000, 10000, 20000 and 50000, respectively. Since Genocop III has a bad performance on the test function No. 23 (SC_6), the results excluding this function are also included.

According to the results in Table 5.7, the performance of DTS_{APS} is generally better than Genocop III and SS when the number of function evaluations is greater than 1000. However, in the early stage of computations, SS performs better than DTS_{APS}. This can be expected since DTS_{APS} is a point-to-point search method while SS is a population-based search method. So, DTS_{APS} may need more iterations, and therefore more function evaluations, to explore the search space well especially for high dimensional functions ($n \geq 6$).

Since the data related to Genocop III and SS in Fig. 5.5 are taken from [59], we also made the successful trial test [59] for the DTS_{APS} results in our experiments. We say that a method approximately finds an optimal solution if

$$GAP \leq \begin{cases} 0.001, & \text{if } f^* \neq 0, \\ 0.001 f^*, & \text{otherewise.} \end{cases}$$

The graphs in Fig. 5.5 show the number of test functions from Set B that were approximately solved by each method. Fig. 5.5 shows that DTS_{APS} could approximately find global minima for 14 test functions within only 100 function evaluations. Actually, the dimensions of these 14 test functions are less than or equal to 6. Fig. 5.5 also shows that DTS_{APS} generally outperforms the other two methods, Genocop III and SS.

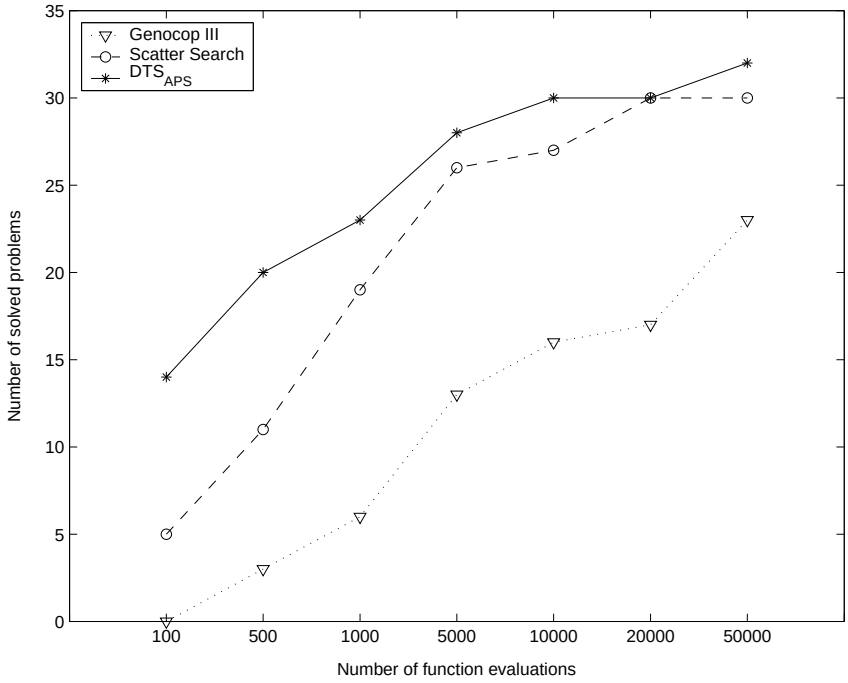

Figure 5.5: Number of solved problems.

5.6 Conclusion

In this chapter, we have presented a continuous TS method called Directed Tabu Search (DTS) method. First, neighborhood-local search strategies are introduced to provide more powerful search procedures to generate trial moves. A new pattern search procedure and the NM method are used to construct these neighborhood-local search strategies. Moreover, new memory elements called *TR*, Semi-*TR* and multi-ranked *TL* are applied to compose anti-cycling rules and to escape from local minima. Finally, a diversification scheme based on the memory element *VRL* is introduced to explore broad areas of the solution space. The numerical results show the promise of the proposed method.

Chapter 6

Filter SA for Constrained Global Optimization

6.1 Introduction

In Chapters 2 and 4, SA-based methods have been proposed to deal with unconstrained global optimization problems. This chapter aims to extend the SA-based methods coverage to constrained global optimization problems. Actually, implementing SA on the continuous constrained optimization problem is still very limited in comparison with some other metaheuristics like the Evolutionary Algorithms (EAs). The SA approaches for constrained global optimization problems have been proposed by Wah, Wang and Chen, see [17, 89, 90, 91]. Another SA approach has been proposed by Romeijn and Smith [82]. These approaches are regarded as a pure SA. In this chapter we propose a hybrid SA approach which invokes some intelligent concepts from other metaheuristics and local search methods. Specifically, we propose a SA-based approach called Filter Simulated Annealing (FSA) method for the constrained optimization problem

$$\begin{array}{ll} \min\limits_{x} & f(x), \\ \text{s.t.} & g_i(x) \leq 0,\ i = 1, \ldots, l, \\ & h_j(x) = 0,\ j = 1, \ldots, m, \\ & x \in \mathcal{S}, \end{array} \tag{P}$$

where f , g_i and h_j are real-valued functions defined on the search space $\mathcal{S} \subseteq R^n$. Usually, the search space $\mathcal{S}$ is defined as $\{x \in R^n : x_i \in [l_i, u_i]\,,\ i = 1, \ldots, n\}$. The feasible region defined by the constraints is denoted by $\mathcal{F} \subseteq \mathcal{S}$.

Most of the metaheuristics which have been proposed to solve problem (P) are Evolutionary Algorithms (EAs). The optimization methods can be classified in two categories; the point-to-point methods which SA belongs to, and the population-based methods which EAs belong to. Metaheuristics from both categories have been successfully applied to the continuous unconstrained optimization problem. However, invoking point-to-point methods to deal with continuous constrained optimization problems is still very limited in comparison with the population-based methods. The main reason for unpopularity of SA for constrained global optimization problems, as well as most of the point-to-point methods, is its difficulty in keeping diversity. Especially, when the feasible region consists of several disjointed sub-regions, it is not so easy for a point-to-point method without a guidance of a diversification scheme to explore such regions effectively. Moreover, the point-to-point methods can be divided in two classes; single-start methods and multi-start methods. The latter methods have shown efficient performance when applied to difficult optimization problems [62, 63, 84]. The standard SA belongs to the class of single-start methods. Therefore, there is a need to modify the standard SA in order to obtain an efficient method that can deal with the general case of problem (P).

In order to compose a powerful point-to-point-based method for solving problem (P), it is highly needed to consider the following things:

- In order to achieve efficient exploration of the space of interest, the designed method should consist of multi-start stages with a guidance of an effective diversification scheme. Otherwise, in the case of having disjointed feasible sub-regions, the method may be trapped in the first hit feasible sub-region.

- An efficient exploration process should also invoke a search procedure which has the ability to explore both of feasible and infeasible regions, rather than exploring the feasible region only. This is needed to reach the global solution specially in the following cases:

  - the global solution lies on the boundary of the feasible region,
  - the global solution lies in a feasible sub-region which differs from the currently considered sub-region in the case of having several disjointed feasible sub-regions.

- In most cases of constrained optimization problems like problem (P), optimal solutions usually lie on the boundary of the feasible region. In order to explore the region near the boundary between the feasible and infeasible regions effectively, the designed method should invoke a solution generation procedure which is able to intensify the solution generation process.

- An elite-based intensification scheme is needed in the final stage in order to refine the best solution found so far. Especially, if the method is SA-based, a quicker intensification scheme is highly needed to overcome the slowness of SA in its final stage.

We have considered all the above in designing the FSA method. So the FSA method is a multi-start method with a diversification scheme. The FSA method uses the filter set concept [24] in accepting new trial solutions, which gives it the ability to explore both of feasible and infeasible regions. Moreover, the FSA method generates more trial solutions whenever the region near the boundary is reached. Finally, two types of intensification schemes are applied in order to refine the best solution visited so far. Thus, the FSA method is a hybrid method which takes advantage of low computational cost of point-to-point methods and efficient exploration of population-based methods. In other words, the FSA method is an attempt to design a point-to-point method that behaves like a population-based method without spending a big computational cost.

The numerical results shown later indicate that the proposed FSA method is very promising in the quality of obtained solutions as well as the computational costs especially for dealing with constraints. Moreover, the numerical results also show that the FSA method is competitive with the population-based methods in the quality of solution and it is much cheaper than them in the computational costs. In the next section, we give some preliminaries needed throughout this chapter. In Section 6.3, we highlight the main components of the proposed FSA method. The study of the FSA parameters is given in Section 6.4. In Sections 6.5 and 6.6, we report numerical results for the FSA method. Finally, the conclusion of the contribution of this chapter makes up Section 6.7.

6.2 Preliminaries

This section highlights the idea of reformulating problem (P) as a multiobjective optimization problem and the concepts of filter set and filtered points. To achieve that, the concept of Pareto dominance in multiobjective optimization should be defined first.

6.2.1 Pareto Dominance

Pareto Dominance is the most common concept of optimality in the multiobjective optimization field. Multiobjective optimization seeks to optimize a vector of objective functions within a feasible decision variable space. For the multiobjective minimization problem with the objective functions $\varphi_1(x), \ldots, \varphi_q(x)$, defined on the search space $\mathcal{S}_M \subseteq R^n$, the Pareto Dominance is defined as follows:

Definition 6.2.1. *An objective vector* $\Phi(y) = (\varphi_1(y), \ldots, \varphi_q(y))$ *is said to dominate another objective vector* $\Phi(z) = (\varphi_1(z), \ldots, \varphi_q(z))$, *written* $\Phi(y) \prec \Phi(z)$, *if and only if* $\varphi_i(y) \leq \varphi_i(z), \forall i = 1, \ldots, q$ *and there exists at least one* $j \in \{1, \ldots, q\}$ *such that* $\varphi_j(y) < \varphi_j(z)$.

We will write $\Phi(y) \preceq \Phi(z)$ to indicate that either $\Phi(y) \prec \Phi(z)$ or $\Phi(y) = \Phi(z)$. In the rest of this chapter, we will simply write $y \prec z$ and $y \preceq z$ instead of writing $\Phi(y) \prec \Phi(z)$ and $\Phi(y) \preceq \Phi(z)$, respectively.

6.2.2 Problem Reformulation

An effective approach to handle constraints is to use multiobjective optimization techniques, see for example [18, 19]. Such approaches reformulate the constrained problem as a multiobjective problem involving the original objective function and constraint violation functions. More specifically, by introducing the constraint violation functions

$$\begin{aligned} G_i(x) &= (\max[0, g_i(x)])^\alpha, \ i = 1, \ldots, l, \\ G_{l+j}(x) &= |h_j(x)|^\alpha, \ j = 1, \ldots, m, \end{aligned} \tag{6.2.1}$$

where α is normally chosen to be 1 or 2, problem (P) can be reformulated as the following multiobjective optimization problem:

$$\min_{x \in \mathcal{S}} [f(x), G_1(x), \ldots, G_{m+l}(x)]. \tag{P_M}$$

Alternatively, we may consider another reformulation of problem (P) as the following biobjective optimization problem:

$$\min_{x \in \mathcal{S}} [f(x), G(x)], \tag{P_B}$$

where $G(x) = \sum_{i=1}^{m+l} G_i(x)$. The method proposed in this chapter will deal with problem (P) through the reformulated problem (P_B). In particular, we will denote $x \prec y$ if x dominates y with respect to the vector function $\Phi(x) = (f(x), G(x))$.

6.2.3 Filter Set and Filtered Points

The *filter set* $\mathfrak{F}$ is defined as a finite set of infeasible[1] points in $\mathcal{S}$ such that $x \prec y$ does not hold for any x and y in $\mathfrak{F}$. The point x^F with the minimum function value $f(x)$ found so far in the feasible region $\mathcal{F} = \{x \in \mathcal{S} : G(x) = 0\}$ is saved and treated separately as a single filter point. This definition is taken from [5] which differs slightly from the original definition in [24]. A point y is called a *filtered point* [5], if one of the following holds:

- $y \succeq x$ for some $x \in \mathfrak{F}$.
- $G(y) \geq G_{\max}$, where $G_{\max} > 0$ is maximum value allowed on the constraint violation function $G(x)$.
- $G(y) = 0$, and $f(y) \geq f^F$, where $f^F = f(x^F)$ is the minimum function value found so far in the feasible region.

In other words, we have three kinds of filtered point sets:

$$\begin{aligned} \bar{\mathfrak{F}}_{\mathrm{I}} &= \{y \in \mathcal{S} : y \succeq x \text{ for some } x \in \mathfrak{F}\}, \\ \bar{\mathfrak{F}}_{\mathrm{II}} &= \{y \in \mathcal{S} : G(y) \geq G_{\max}\}, \\ \bar{\mathfrak{F}}_{\mathrm{III}} &= \left\{y \in \mathcal{S} : G(y) = 0,\ f(y) \geq f^F\right\}. \end{aligned}$$

Therefore, the set of all filtered points is defined as $\bar{\mathfrak{F}} = \bar{\mathfrak{F}}_{\mathrm{I}} \cup \bar{\mathfrak{F}}_{\mathrm{II}} \cup \bar{\mathfrak{F}}_{\mathrm{III}}$. Unfiltered points are used to update $\bar{\mathfrak{F}}$ by adding them and deleting the old ones which are dominated by the new added points.

6.3 The FSA method

The FSA method starts with a diversification generation procedure to generate a set of diverse solutions called *DivSet*. The initial solution is chosen from the *DivSet*. Then, the *DivSet* stands by to provide the search with a diverse solution whenever further diversification is needed. In the FSA method, we introduce a ranking procedure for comparing and ordering solutions. This ranking procedure is based on the filter set as well as objective function and constraint violation function values. The scenario in the FSA method can be

[1] Throughout this chapter, the feasibility is related only to problem (P) rather than P_M or P_B, that is, we call a point $x \in \mathcal{S}$ feasible if $G(x) = 0$, and infeasible if $G(x) > 0$.

described as follows. Let the current trial solution be *Sol*. Using the FSA ranking procedure, *Sol* is initialized to be the best ranked one in *DivSet*. Then, trial solutions are generated in a neighborhood of *Sol* using a trial solution generation procedure based on the approximate descent direction (ADD) method proposed in Chapter 4. The trial solution generation procedure generates trial solutions in such a way that the objective function value is likely to decrease if *Sol* is feasible, and the constraint violation function value is likely to decrease if *Sol* is infeasible. Moreover, the trial solution generation procedure intensifies the solution generation process if *Sol* is close to the boundary of the feasible region. We try to update *Sol* with one of the generated trial solutions using the simulated annealing acceptance concept. Specifically, if an unfiltered trial solution is obtained, we accept it with probability 1. Otherwise, a trial solution is accepted with a certain probability controlled by the temperature parameter. Whenever the number of consecutive iterations without accepting a new trial solution exceeds a predetermined maximum number, a new diverse solution is chosen from *DivSet* and the re-annealing process is applied, i.e., the temperature is re-initialized. While the search proceeds, *DivSet* is updated by removing any of its elements if the search reaches a region close to this element. We terminate this main stage of the FSA method when the cooling schedule is completed with an empty *DivSet*. Finally, two intensification schemes are invoked to refine the best solution found so far. The best solution is defined to be the best feasible solution if the feasible region is reached. Otherwise, the best solution is defined to be the infeasible solution with the least constraint violation function value. The temperature parameter at the best solution found so far in the previous search stage is saved to be used in the first intensification scheme which applies an annealing process with slower cooling schedule and smaller step sizes. The second intensification scheme applies a greedy local search method on a penalty function of problem (P) starting from the best solution found so far. Figure 6.1 shows the outline of the FSA method. Below, we describe the details of the FSA main steps sketched above and state the FSA algorithm formally at the end of this section.

6.3.1 Diversification Generation Procedure

In the FSA method, we use the scatter search diversification generation Method [59, 60] to generate a diverse solution set *DivSet.* In that method, the interval $(u_i - l_i)$ of each variable is divided into 4 sub-intervals of equal size. For each sub-interval of each variable, a *frequency count* is defined as the number of solutions which are perviously chosen in this sub-interval. To generate a new solution to be added to *DivSet*, one has to

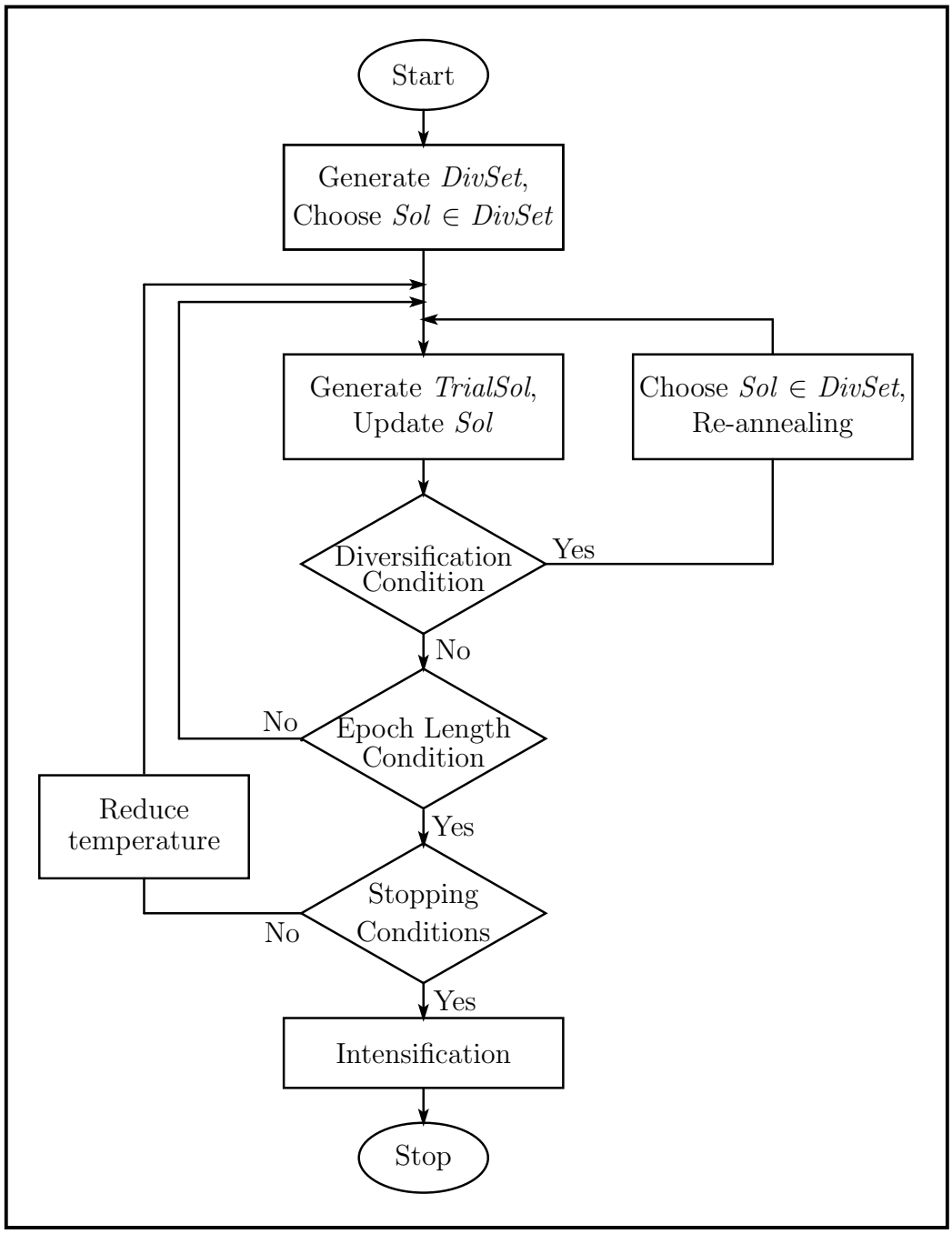

Figure 6.1: Outline of the FSA method.

- choose one sub-interval for each variable randomly with a probability inversely proportional to its frequency count, and
- choose a random value for each variable that lies in the corresponding selected sub-interval.

While the search proceeds, the *DivSet* is updated by eliminating any of its elements lying close to a visited solution. Specifically, when the current solution is x, the *DivSet* is updated through the rule

$$DivSet = DivSet - \{y : y \in DivSet,\ \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{H_i^2} \leq 1\}, \tag{6.3.1}$$

where $H_{Div} = (H_1, \ldots, H_n)$ is a predetermined constant vector with positive components.

When the diversification is needed, the solution with the largest distance from the current solution is chosen from the *DivSet* to be a new diverse solution.

6.3.2 Ranking Procedure

To order the solutions in a set $S = \{x_1, x_2, \ldots, x_\mu\}$, we introduce the following ranking procedure. The solutions are ordered based on three rank functions as given below.

1. *Dominance Rank* (r_d): The best feasible point x^F is given the rank value $r_d = 1$, and other feasible points are given the rank value $r_d = 2$. The points in $\mathfrak{F}$ are given the rank value $r_d = 1$, and any other infeasible point x is given the rank value $r_d = \nu + 1$, where ν is the number of points in $\mathfrak{F}$ which dominate x.
2. *f-value Rank* (r_f): According to their objective function values $f(x_i)$, $x_i \in S$, the best point is given the rank value $r_f = 1$, the second best point is given the rank value $r_f = 2$, and so on.
3. *G-value Rank* (r_G): According to their constraint violation function values $G(x_i)$, $x_i \in S$, the best point is given the rank value $r_G = 1$, the second best point is given the rank value $r_G = 2$, and so on.

In each ranking described above, ties are broken arbitrarily. Then, the total ranking function r is defined by

$$r(x_i) = r_d(x_i) + \frac{\lambda}{\mu} r_f(x_i) + \frac{(1-\lambda)}{\mu} r_G(x_i),\ x_i \in S, \tag{6.3.2}$$

where $\lambda \in [0,1]$. The solutions of S are ordered and relabeled such that

$$r(x_1) \leq r(x_2) \leq \cdots \leq r(x_{n+1}). \tag{6.3.3}$$

The main role of the parameter λ is to control the priority in the ranking between the objective function value and the feasibility. Actually, the ranking function r is basically based on the dominance rank r_d and, within the same dominance rank value, the parameter λ gives a greater value to either of the ranking values r_f and r_G. Specifically, setting $\lambda \in [0, 0.5)$ gives some priority to the feasible points and setting $\lambda \in (0.5, 1]$ gives some priority to points with lower objective function values. In the FSA method, the value of λ is chosen to be less than $1/\mu$ in order to accept a better feasible solution when it is found. Moreover, this ranking procedure allows a new infeasible solution which reduces the constraint violation function to be highly accepted, compared with a new feasible solution which is worse than the best feasible solution found so far. This gives the search process more flexibility to explore the boundary region.

6.3.3 Trial Solution Generation Procedure

We use the Approximate Descent Direction (ADD) method proposed in Chapter 4 to generate trial solutions in the FSA method. Specifically, we use the ADD method to generate a search direction d at a given solution x, and then use it to generate new trial solutions in a neighborhood of x. To achieve that, we first generate p exploring points $\{y_i\}_{i=1}^p$ close to x and generate the search direction d as follows:

1. If x is feasible, we apply the ADD method with $\{y_i\}_{i=1}^p$ and (4.2.1) to compute an approximate descent direction v of f at x. Then, we set the search direction $d := v/\|v\|$.

2. If x is infeasible, we apply the ADD method with $\{y_i\}_{i=1}^p$ and use (4.2.1) to compute an approximate descent direction v of G at x. Then, we set the search direction $d := v/\|v\|$.

Trial solutions can be generated along the search direction d with suitable step sizes. Moreover, it is known that, in most cases, optimal solutions can be found on the boundary of the feasible region. So, in order to encourage the search to explore the region near the boundary effectively, more trial solutions should be generated whenever the current solution is close to the boundary. To implement this idea in the FSA method, another trial solution

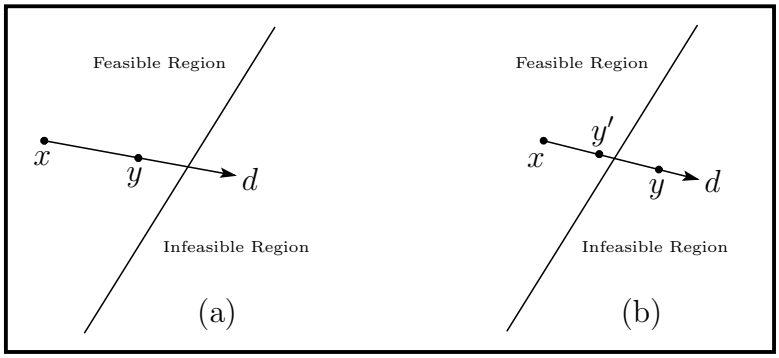

Figure 6.2: An example of generating trial solutions.

will be generated between the current solution and the trial solution if the feasibility status is changed between them. Figure 6.2 shows an example of the two types of generating trial solutions in the neighborhood of a current solution x. In Figure 6.2(a), a trial solution y is generated along the search direction d and since x and y are feasible, no more trial solution will be generated. However, in Figure 6.2(b), x is feasible but y is infeasible, and so another trial solution y' is generated between x and y. Formally, we can define the trial solution set as

$$TS(x) = \{y : y = x + \delta_i \Delta d,\ i \in I\}, \tag{6.3.4}$$

where Δ is a step size and δ_i are random numbers. The set I is given by $I = \{1\}$ if the feasibility status at $x + \delta_1 \Delta d$ is the same as that at x, i.e., $G(x + \delta_1 \Delta d) = G(x) = 0$, or $G(x + \Delta d) > 0$ and $G(x) > 0$, and $I = \{1, 2\}$, otherwise. The random numbers δ_i give the search some stochastic behavior to achieve more efficient exploration. For example, we may let δ_1 be uniformly distributed in the interval $(0, 1)$ and δ_2 be normally distributed with mean $1/2$ and a suitable variance σ^2.

6.3.4 Intensification

In the FSA method, we compose two stages of intensification process. The first one is SA-based procedure, called *SA Intensification*, in which up-hill movements may be accepted in order to avoid the case where the region around the best solution visited so far is prematurely explored in the previous search stages. The other stage of intensification is a greedy process that accepts only down-hill movements, which we call *Local Search Intensification.* This greedy-type intensification is needed since it has been reported that the SA can reach a region near global minima, however, it may wander around the optimal solution if high

accuracy is required [41, 92]. The outline of these intensification stages is given below.

- **SA Intensification.** In the previous stage of the search, we save the temperature parameter value recorded at the best solution found so far. Then, in order to refine that solution, a slower cooling schedule, i.e., a schedule with a higher cooling ratio, is started from the saved value of the temperature parameter. Moreover, the step size used in generating trial solutions is reduced to refine the search steps for more accurate exploration.

- **Local Search Intensification.** A direct search method is applied, starting from the best solution found so far, to minimize the penalty function

$$p(x) = f(x) + \rho G(x), \tag{6.3.5}$$

where ρ is a penalty parameter. The Kelley's modification [51, 52] of the Nelder-Mead method [72] is used to minimize the function $p(x)$ in N consecutive times using gradually increasing penalty parameters $\rho_1, \rho_2, \ldots, \rho_N$.

6.3.5 FSA Algorithm

The formal description of the FSA method is given below.

Algorithm FSA

1. Initialization. Construct *DivSet* using the diversification generation procedure. Set the best ranked point in *DivSet* to be the initial point x_0. Choose the cooling schedule parameters; initial temperature T_{max}, final temperature T_{min} and cooling ratio $\gamma \in (0, 1)$, and the epoch length M and set $T := T_{max}$. Set $\mathfrak{F}_0$ to be empty, set $x^{best} = x_0$, choose a step size $\Delta > 0$, choose a positive integer $K_{\max}$, and set $k := 0$.

2. Main Loop.

2.1. Compute a trial solution set $TS(x_k)$ as in (6.3.4) with the step size Δ. Set y_k equal to the best ranked point in $TS(x_k)$.

2.2. The trial point y_k is accepted with the probability

$$p = \begin{cases} 1, & \text{if } y_k \notin \bar{\mathfrak{F}}_k, \\ \min\{1, \exp\left(-\Delta_{fG}/T\right)\}, & \text{otherwise}, \end{cases} \tag{6.3.6}$$

where $\Delta_{fG} = \max\left\{f\left(y_k\right) - f\left(x_k\right), G\left(y_k\right) - G\left(x_k\right)\right\}$.

2.3. If y_k is accepted, then set $x_{k+1} := y_k$; otherwise, set $x_{k+1} := x_k$. Update $\mathfrak{F}_k$, x^{best} and *DivSet*, and set $k := k + 1$.

2.4. *Diversification.* If the number of consecutive iterations without accepting a new solution exceeds $K_{\max}$, and *DivSet*$\neq \phi$, then choose $x_k \in$ *DivSet*, set $T := T_{max}$, set $\mathfrak{F}_k$ to be empty, and go to Step 2.1. Otherwise, go to Step 2.5.

2.5. If the epoch length M is attained, then go to Step 2.6. Otherwise, go to Step 2.1.

2.6. If $T > T_{min}$, then set $T := \gamma T$ and go to Step 2.1. Otherwise, go to Step 3.

3. Intensification.

3.1. *SA Intensification.* Set x_k equal to x^{best}, set T equal to the saved temperature at that point, and set a final temperature T'_{min}, an epoch length M' and a cooling ratio $\gamma' > \gamma$.

3.1.1 Compute a trial solution set $TS(x_k)$ as in (6.3.4) with the step size Δ. Set y_k equal to the best ranked point in $TS(x_k)$.

3.1.2 Accept y_k with the probability p given by (6.3.6). Set $x_{k+1} := y_k$ if y_k is accepted; otherwise, set $x_{k+1} := x_k$. Update $\mathfrak{F}_k$ and x^{best}, and set $k := k + 1$.

3.1.3 If the epoch length M' is attained, then go to Step 3.1.4. Otherwise, go to Step 3.1.1.

3.1.4 If $T > T'_{min}$, then set $T := \gamma' T$ and go to Step 3.1.1. Otherwise, go to Step 3.2.

3.2. *Local Search Intensification.* For $\rho = \rho_1, \rho_2, \ldots, \rho_N$:

3.2.1 Apply a local search method to the function $f(x) + \rho G(x)$ starting from x^{best}.

3.2.2 Update x^{best} and go to Step 3.2.1.

6.4 Setting FSA Parameters

In this section, setting the FSA parameters is discussed to complete the implementation of the FSA algorithm stated in the previous section. These parameters can be classified as shown Table 6.1, which contains all FSA parameters and their definitions. Some preliminary numerical experiments have been done in order to find proper values of these parameters. Moreover, these experiments of tuning parameters aim to obtain a standard setting of parameters which is problem-independent as much as possible. The values of some parameters

Table 6.1: The FSA parameters

Parameter Group	Parameter	Definition
Constraint Violation Function	α	Power factor used in (6.2.1)
	ϵ	Small positive number used for reformulating equality constraints
Diversification	$\|DivSet\|$	Size of *DivSet*
	H_{Div}	Distance vector used to update *DivSet*
	$K_{\max}$	Maximum number of iterations allowed without acceptance
Cooling Schedule	T_{max}, T_{min}	Initial and final temperatures
	M	Epoch length
	γ	Cooling ratio
Trial Solutions	p	Number of exploring points used in ADD
	r	Neighborhood radius used in ADD
	Δ	Step size used in (6.3.4)
	σ^2	Variance of the normal distribution of δ_2
	λ	Rank ordering parameter
	$G_{\max}$	Maximum value allowed on $G(x)$
Intensification	T'_{min}	Final temperature in SA Intensification
	M'	Epoch length
	γ'	Cooling ratio in SA Intensification
	$\rho_1, \rho_2, \ldots, \rho_N$	Penalty parameters

are assigned to their standard setting reported in the literature. Below, we state the suggested values of the FSA parameters as well as the conclusion of what we got from the tuning parameters experiments.

6.4.1 Constraint Violation Function Parameters

The power factor α used in (6.2.1) is set equal to 2, since using this value showed notably better performance of the FSA method than that of using the value 1. Treating the equality constraints as in (6.2.1) does not seem efficient in the implementation. It was observed that reformulating the equality constraint $h(x) = 0$ as the inequality constraint $|h(x)| - \epsilon \leq 0$, where ϵ is a small positive number, yielded a better performance of the FSA method. Moreover, using a large value of ϵ in the first stage of search and reducing its value in the intensification stage gave better results. Therefore, we set ϵ equal to 10^{-3} in reformulating

all equality constraints in all FSA search stages except in the local search intensification stage in which ϵ is set equal to 10^{-6}.

6.4.2 Diversification Parameters

The size of *DivSet* depends on many factors such as the wideness of the search space, the number of disjoint feasible sub-regions, and the multimodality of the objective function. We observed that setting the size of *DivSet* equal to 50 almost fits all the considered problems. The distance vector $H_{Div} = (H_1, \ldots, H_n)$ used to update the *DivSet* is set so as to fit the size of the search space. Specifically, we set $H_i = \frac{u_i - l_i}{|DivSet|/n}$, $i = 1, \ldots, n$, where the denominator represents the average line density of distributing the solutions of *DivSet* along each coordinate direction, so that the value of H_i represents the average distance along the coordinate direction i between two neighboring diverse solutions. The maximum number $K_{\max}$ of iterations allowed without accepting new trial solutions is set equal to 10.

6.4.3 Cooling Schedule Parameters

The initial temperature $T_{\max}$ is set large enough to make the initial probability of accepting transition close to 1. Beside the initial point x_0, another point $\widetilde{x}_0$ is generated in a neighborhood of x_0 to calculate $T_{\max}$ as

$$T_{\max} := -\frac{1}{\ln(0.9)} \left| f(\widetilde{x}_0) - f(x_0) \right| .$$

At the beginning of each re-annealing process, a new $T_{\max}$ is computed in a similar manner. The cooling ratio γ is normally chosen to be between 0.9 and 0.99 [57]. In our experiments, we set γ equal to 0.9 and a higher value is used in the intensification stage as we will state later. A common choice of the epoch length M is to let it depend on the size of the problem [53, 58]. In our experiments, we set M equal to $2n$. The cooling schedule is terminated when the temperature reaches a fixed minimum value $T_{\min}$. We observed that setting $T_{\min}$ equal to $\min\left(10^{-5}, 10^{-5} T_{\max}\right)$ could give a complete cooling schedule in the sense that the acceptance probability at the end is almost zero.

6.4.4 Trial Solutions Parameters

The parameters used in computing the search directions v_f and v_G are the number p of exploring points, and the radius r of the neighborhood in which the exploring points are

generated. We set $p = 2$ and $r = 10^{-3}$ as suggested in Chapter 4. The ranking parameter λ is set equal to $0.5/\mu$, where μ is the number of solutions to be ranked or compared. This setting allows the best feasible solution to have the highest rank, whenever it exists, among the compared solutions. Setting a proper value of step size Δ is very effective in the performance of the FSA algorithm, because setting too big a value for Δ may yield a premature termination of the algorithm and setting too small a value for Δ will not yield an efficient exploration process for the whole search space. We tested many values of Δ and found that the value $\Delta = \min\left(0.05\sum_{i=1}^{n}(u_i - l_i)/n, 10\right)$ gave the best performance. For variance σ^2 of the normal distribution of δ_2, the values $\sigma = 1/2, 1/3, 1/4$ have been tested. We observed that setting $\sigma = 1/3$ gave a slightly better performance than setting the other values. The filter set contains only one parameter; the maximum value $G_{\max}$ allowed on the constraint violation function $G(x)$. In the original reference of the filter set concept [24], the value of $G_{\max}$ is set equal to $\max(1.25G(x_0), 100)$, where x_0 is the initial solution. However, in the FSA algorithm, we use a higher value, since our goal is to explore the whole search space effectively and reaching a global minimum, which differs from the goal of [24], i.e., finding a local minimum. So we set $G_{\max}$ equal to $10\max(1.25G_{\max}^{Div}, 100)$, where $G_{\max}^{Div}$ is the maximum value of the constraint violation function $G(x)$ computed at each point in the *DivSet*.

6.4.5 Intensification Parameters

For the SA Intensification Parameters, final temperature T'_{min}, epoch length M' and cooling ratio γ' are set equal to $10^{-5}T_{best}$, $2n$ and 0.99, respectively, where T_{best} is the temperature saved at the best solution found so far. The number N of times the local search method is applied in the local search intensification stage is set equal to 4. The penalty parameters used in these local searches are $\rho_1 = 10^{\beta+2}$, $\rho_1 = 10^{\beta+4}$, $\rho_1 = 10^{\beta+6}$ and $\rho_1 = 10^{\beta+10}$, where β is the number that appears in the floating point form $\alpha_1.\alpha_2\alpha_3\cdots\times 10^{\beta}$ of the best point found so far.

6.5 Numerical Results

In this section, we report the performance of the FSA algorithm on 13 well-known test problems G1–G13 [45, 55, 69], which are shown in Appendix B. The characteristics of those test problems are diverse enough to cover many kinds of difficulties that constrained global

optimization problems face. More experimental results on three other application problems will be shown in the next section.

The FSA code was applied to solve each considered problem 30 times with different starting solutions. For all test problems, the values of the FSA parameters remained constant at those values which have been presented in the previous section. Table 6.2 summarizes the FSA results obtained for each test problem as well as the best known objective function value for each problem. Problems G2, G3 and G8 are maximization problems originally so that they were solved by converting them to minimization problems. In Table 6.2, the best and the worst objective function values obtained from 30 runs are reported for each test problem. In order to show more details concerning the quality of the obtained solutions, the average and the standard deviation of the obtained objective function values are also reported in Table 6.2. Moreover, the average numbers Av. f-evals. and Av. c-evals. of objective and constraint functions evaluations, respectively, are shown in the last two columns of Table 6.2. It is noteworthy that the FSA method is very economical in computing the constraint function values as shown in Table 6.2.

The results obtained by the FSA method are quite satisfactory, except for problem G2 which has the highest dimension among all test problems G1–G13. On the other hand, the results for problem G12 are very promising since the feasible region of this problem consists of 9^3 disjointed spheres with radus 0.25. The FSA method could successfully find global minima in all runs with low computational costs as shown in Table 6.2. This indicates the success of the multi-start diversification scheme invoked in the FSA method. For problem G11, the FSA method reached a point with objective function value 0.7499990 for all 30 runs. However, by decreasing the parameter ϵ, which is used to convert the equality constraint to the inequality one, from 10^{-6} to 10^{-10} in the local search intensification, the FSA method easily reached the exact global minimum with objective function value 0.75 in all runs.

To complete examining the FSA performance, its results are compared with those of other EA-based methods proposed for dealing with problem (P). The EA-based methods that we used in the comparison are

1. Homomorphous Mappings (HM) method [55],
2. Stochastic Ranking (SR) method [83],
3. Adaptive Segregational Constraint Handling EA (ASCHEA) method [36],
4. Simple Multimembered Evolution Strategy (SMES) method [66].

Table 6.2: FSA results for problems G1–G13

Pr.	Type	Best Known	Best	Av.	Worst	S.D.	Av. f-evals.	Av. c-evals.
G1	min	–15	–14.999105	–14.993316	–14.979977	0.004813	205,748	87,701
G2	max	0.803619	0.7549125	0.3717081	0.2713110	0.098023	227,832	101,903
G3*	max	1	1.0000015	0.9991874	0.9915186	0.001653	314,938	118,404
G4	min	–30665.539	–30665.5380	–30665.4665	–30664.6880	0.173218	86,154	37,000
G5*	min	5126.4981	5126.4981	5126.4981	5126.4981	0.000000	47,661	17,757
G6	min	–6961.81388	–6961.81388	–6961.81388	–6961.81388	0.000000	44,538	15,817
G7	min	24.3062091	24.310571	24.3795271	24.644397	0.071635	404.501	171,299
G8	max	0.095825	0.095825	0.095825	0.095825	0.000000	56,476	23,219
G9	min	680.6300573	680.63008	680.63642	680.69832	0.014517	324,569	147,035
G10	min	7049.3307	7059.86350	7509.32104	9398.64920	542.3421	243,520	93,667
G11*	min	0.75	0.7499990	0.7499990	0.7499990	0.000000	23,722	8,485
G12	min	–1	–1.0000000	–1.0000000	–1.0000000	0.000000	59,355	25,818
G13*	min	0.0539498	0.0539498	0.2977204	0.4388511	0.188652	120,268	42,268

* Problems contain equality constraints.

The challenge that the FSA method faces is to what extent a point-to-point method behaves like a population-based method or even better. To examine this issue, two measurements are considered; solution qualities and computational costs. First, we discuss the solution qualities and, later at the end of this section, we will discuss computational costs. The results of the compared methods, which are taken from their original references [36, 55, 66, 83], as well as those of the FSA method are reported in Table 6.3 to show the solution qualities obtained by them. It is not easy to draw a definite conclusion from comparisons due to different accuracies used in the respective results. However, we state below some comments on the results reported in Table 6.3. All the results in Table 6.3 are obtained from 30 runs of each method except those of HM method, which are obtained from 20 runs. The HM method could obtain the optimal solution in all runs for only problem G11. For the other methods SR, ASCHEA, and SMES, they could obtain the optimal solutions in all runs for problems {G1,G3,G4,G8,G11,G12}, {G4,G6,G8,G11} and {G1,G4,G8,G12}, respectively. The FSA method could obtain the optimal solutions in all runs for problems {G5,G6,G8,G11,G12}. It is noteworthy that the FSA method could obtain the optimal solution in all runs for problem G5, whereas the other methods failed to obtain it even in a single run. Moreover, the FSA method could obtain the optimal solution of problem G13 in 7 out of 30 runs, while the other methods failed to obtain it.

The computational costs of the considered EA-based methods are extremely expensive compared with those of the FSA method. Since there is no automatic termination criteria

for the considered EA-based methods, thy were terminated when the number of generations exceeds a predetermined maximum number. Therefore, the computational costs of these methods are problem-independent, i.e., the number of objective and constraint functions evaluation remains constant for each test problem. Specifically, computational costs for HM, SR, ASCHEA and SMES used in each test problem, which are taken from their original references [36, 55, 66, 83], are 1400000, 350000, 1500000 and 250000 fitness function evaluations, respectively, and each fitness function evaluation requires one evaluation of the objective function and one evaluation of each constraint function. The main reason for these high computational costs is that EAs are not equipped with automatic termination criteria and this is one main drawback of EAs. For some of the test problems, the considered EA-based methods could obtain an optimal solution in an early stage of the search, but they were not learned enough to judge whether they could terminate. On the other hand, the EA-based methods have less parameters than SA-based methods. However, in the FSA method as well as SA-based methods, some preliminary experiments on tuning parameters will let them learn applicable termination criteria.

6.6 More Numerical Experiments

In this section, we discuss the results of the FSA method on some application problems. Three problems from the engineering optimization area are considered.

6.6.1 Welded Beam Design Problem

The welded beam design problem [19, 22] yields an optimization problem which has four design variables $x = (x_1, x_2, x_3, x_4)$ and takes the following form:

$$\begin{aligned}
\min_x \ & f(x) = 1.10471 x_1^2 x_2 + 0.04811 x_2 x_2 (14 + x_2), \\
\text{s.t.} \ & g_1(x) = \tau(x) - 13000 \leq 0, \\
& g_2(x) = \sigma(x) - 30000 \leq 0, \\
& g_3(x) = x_1 - x_2 \leq 0, \\
& g_4(x) = 6000 - P_c(x) \leq 0, \\
& g_5(x) = \delta(x) - 0.25 \leq 0, \\
& 0.125 \leq x_1 \leq 10, \ 0.1 \leq x_2, x_2, x_2 \leq 10,
\end{aligned}$$

Table 6.3: Results of FSA and other EA-based methods for problems G1–G13

Pr.	Type	Best Known		HM	SR	ASCHEA	SMES	FSA
G1	min	–15	Best	–14.7864	–15	–15	–15	–14.999105
			Av.	–14.7082	–15	–14.84	–15	–14.993316
			Worst	–14.6154	–15	N.A.	–15	–14.979977
G2	max	0.803619	Best	0.79953	0.803515	0.785	0.803601	0.7549125
			Av.	0.79671	0.781975	0.59	0.785238	0.3717081
			Worst	0.79119	0.726288	N.A.	0.751322	0.2713110
G3*	max	1	Best	0.9997	1.000	1	1.001038	1.0000015
			Av.	0.9989	1.000	0.99989	1.000989	0.9991874
			Worst	0.9978	1.000	N.A.	1.000579	0.9915186
G4	min	–30665.539	Best	–30664.5	–30665.539	–30665.5	–30665.539062	–30665.5380
			Av.	–30655.3	–30665.539	–30665.5	–30665.539062	–30665.4665
			Worst	–30645.9	–30665.539	N.A.	–30665.539062	–30664.6880
G5*	min	5126.4981	Best	–	5126.497	5126.5	5126.599609	5126.4981
			Av.	–	5128.881	5141.65	5174.492301	5126.4981
			Worst	–	5142.472	N.A.	5304.166992	5126.4981
G6	min	–6961.81388	Best	–6952.1	–6961.814	–6961.81	–6961.813965	–6961.81388
			Av.	–6342.6	–6875.940	–6961.81	–6961.283984	–6961.81388
			Worst	–5473.9	–6350.262	N.A.	–6961.481934	–6961.81388
G7	min	24.3062091	Best	24.620	24.307	24.3323	24.326715	24.310571
			Av.	24.826	24.374	24.6636	24.474926	24.3795271
			Worst	25.069	24.642	N.A.	24.842829	24.644397
G8	max	0.095825	Best	0.0958250	0.095825	0.09582	0.095826	0.095825
			Av.	0.0891568	0.095825	0.09582	0.095826	0.095825
			Worst	0.0291438	0.095825	N.A.	0.095826	0.095825
G9	min	680.6300573	Best	680.91	680.630	680.630	680.631592	680.63008
			Av.	681.16	680.656	680.641	680.643410	680.63642
			Worst	683.18	680.763	N.A.	680.719299	680.69832
G10	min	7049.3307	Best	7147.9	7054.316	7061.13	7051.902832	7059.86350
			Av.	8163.6	7559.192	7497.434	7253.047005	7509.32104
			Worst	9659.3	8835.655	N.A.	7638.366211	9398.64920
G11*	min	0.75	Best	0.75	0.750	0.75	0.749090	0.7499990
			Av.	0.75	0.750	0.75	0.749358	0.7499990
			Worst	0.75	0.750	N.A.	0.749830	0.7499990
G12	min	–1	Best	–0.999999857	–1.000000	N.A.	–1.000000	–1.000000
			Av.	–0.999134613	–1.000000	N.A.	–1.000000	–1.000000
			Worst	–0.991950498	–1.000000	N.A.	–1.000000	–1.000000
G13*	min	0.0539498	Best	N.A.	0.053957	N.A.	0.053986	0.0539498
			Av.	N.A.	0.057006	N.A.	0.166385	0.2977204
			Worst	N.A.	0.216915	N.A.	0.468294	0.4388511

* Problems contain equality constraints.

Table 6.4: Results for the welded beam design problem

Method	Best	Av.	Worst	S.D.	Av. f-evals.	Av. c-evals.
GA [19]	1.728226	1.792654	1.993408	0.074713	80,000	80,000
FSA	1.7250022	1.7564428	1.8843960	0.0424175	58,238	24,971

where

$$\tau(x) = \sqrt{(\tau_1(x))^2 + (\tau_2(x))^2 + \frac{x_2\tau_1(x)\tau_2(x)}{\sqrt{0.25[x_2^2+(x_1+x_2)^2]}}},$$
$$\tau_1(x) = \frac{6000}{\sqrt{2}x_1x_2},\ \tau_2(x) = \frac{6000(14+0.5x_2)\sqrt{0.25[x_2^2+(x_1+x_2)^2]}}{2[0.707x_1x_2(x_2^2/12+0.25(x_1+x_2)^2)]},$$
$$\sigma(x) = \frac{504000}{x_2^2x_2},\ P_c(x) = 64746.022(1-0.0282346x_2)x_2x_2^3,\ \delta(x) = \frac{2.1952}{x_2^3x_2}.$$

This problem has been well studied, see [19, 22] and references therein. However, the FSA method could obtain a new solution for it which is better than the one known in the literature. Specifically, the FSA method obtained the solution

$$x^* = (0.20564426101885, 3.47257874213172, 9.03662391018928, 0.20572963979791)$$

with objective function value 1.7250022, while the known solution has the objective function value 1.728226 as reported in [19]. Moreover, the performance of the FSA method is compared against the GA-based method [19] which found the previously known solution. The best, the average, the worst and the standard deviation of objective function values obtained by 30 runs of both methods are reported in Table 6.4. Moreover, the average numbers Av. f-evals. and Av. c-evals. of objective and constraint functions evaluations, respectively, are also shown in Table 6.4. The results related to the GA-based method are taken from the original reference [19]. The figures shown in Table 6.4 indicate the superior performance of the FSA method.

Table 6.5: Results for the pressure vessel design problem

Method	Best	Av.	Worst	S.D.	Av. f-evals.	Av. c-evals.
GA [19]	6059.946341	6177.253268	6469.322010	130.929702	80,000	80,000
FSA	5868.764836	6164.585867	6804.328100	257.473670	108,883	49,253

6.6.2 Pressure Vessel Design Problem

The optimization problem derived from the pressure vessel design problem [19] has four design variables $x = (x_1, x_2, x_3, x_4)$. This problem can be stated as follows:

$$\begin{aligned} \min_x \ & f(x) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3, \\ \text{s.t.} \ & g_1(x) = -x_1 + 0.0193x_3 \le 0, \\ & g_2(x) = -x_2 + 0.00954x_3 \le 0, \\ & g_3(x) = -\pi x_3^2 x_4 - \tfrac{4}{3}\pi x_3^3 + 1296000 \le 0, \\ & g_4(x) = x_4 - 240 \le 0. \end{aligned}$$

The FSA code was run 30 times to solve this problem and the obtained results are summarized in Table 6.5. The results contain the best, the average, the worst and the standard deviation of objective function values, and the average numbers of objective and constraint functions evaluations. The corresponding results of the GA-based method in Table 6.5 are taken from the original reference [19]. The FSA method could obtain a better solution for this problem at

$$x^* = (0.768325709391, 0.379783796302, 39.809622248187, 207.225559518596)$$

with the objective function values 5868.764836.

6.6.3 Tension-Compression String Problem

The problem of minimizing the weight of a tension-compression string [19] can be expressed as the following optimization problem with three design variables $x = (x_1, x_2, x_3)$:

$$\begin{aligned} \min_x \ & f(x) = x_1^2 x_2 (x_3 + 2), \\ \text{s.t.} \ & g_1(x) = 1 - \tfrac{x_2^3 x_3}{71{,}785x_1^4} \le 0, \\ & g_2(x) = \tfrac{4x_2^2 - x_1x_2}{12{,}566x_1^3(x_2 - x_1)} + \tfrac{1}{5{,}108x_1^2} - 1 \le 0, \\ & g_3(x) = 1 - \tfrac{140.45x_1}{x_3x_2^2} \le 0, \\ & g_4(x) = \tfrac{x_1 + x_2}{1.5} - 1 \le 0. \end{aligned}$$

Table 6.6: Results for the tension-compression string problem

Method	Best	Av.	Worst	S.D.	Av. f-evals.	Av. c-evals.
GA [19]	0.012681	0.012742	0.012973	0.000059	80,000	80,000
FSA	0.012665285	0.012665299	0.012665338	0.000000022	49,531	18,802

The FSA code was called 30 times with different starting solutions in order to examine the performance of the FSA method. The results obtained in all runs, as well as those of the GA-based method [19], are reported in Table 6.6. The results of the GA-based method are borrowed from the original reference [19]. The FSA method could obtain the better solution

$$x^* = (0.05174250340926, 0.35800478345599, 11.21390736278739)$$

with the objective function value 0.012665285. The figures in Table 6.6 show that the results obtained by the FSA method are stable for this problem. Moreover, the worst solution obtained by the FSA method is still better than the best one obtained by the GA-based method [19]. Finally, the computational costs of the FSA method are much cheaper than those of the GA-based method [19].

6.7 Conclusion

The hybrid multi-start point-to-point FSA method is proposed. The structure of the FSA method stands on simulated annealing, filter set concept, a new solution generation procedure, and diversification and intensification schemes. These strategies are hybridized in the FSA method in such a way that a point-to-point method behaves like a population-based method without spending much computational cost. The computational results for 13 well-known test problems as well as three application problems are shown to demonstrate the efficiency of the FSA method. A superior behavior of the proposed method against population-based methods in saving the computational costs especially for the constraint function evaluations has been observed.

Chapter 7

Summary and Conclusions

In this study, the continuous global optimization problems in their two forms; unconstrained and constrained problems, have been considered. Derivative-free hybrid methods that combine metaheuristics and direct search methods have been proposed to deal with these problems.

For the unconstrained global optimization problems, four main global search methods have been proposed in Chapters 2–5 based on simulated annealing, genetic algorithm and tabu search. Moreover, direct search methods based on Nelder-Mead, multidirectional search and pattern search methods as well as new proposed methods have been invoked in the four main global search methods in order to overcome the drawbacks of metaheuristics. The numerical results shown in Chapters 2–5 lead to the following remarks:

- Creating direct-search-based logical movements while applying metaheuristics in the proposed methods give better performance of metaheuristics.
- Accelerating the final stage of metaheuristics by applying a complete local search method extricates metaheuristics from wandering around the optimal solution. In other words, applying a complete local search method in the final stage of metaheuristics helps them to obtain good accuracy quickly.
- The proposed methods are promising in practice and competitive with the other compared methods in terms of computational costs and the success of obtaining the global solutions.
- The proposed methods show a superior performance in terms of the solution qualities against the compared methods.

In Chapter 6, SA-based method has been proposed as a hybrid method that combines specific strategies to fit the constrained global optimization problems. Moreover, its computational results shown in Chapter 6 lead to the following remarks:

- A guidance of an effective diversification scheme helps to achieve an efficient exploration of the search domain especially in the case of having disjointed feasible sub-regions.
- An elite-based intensification scheme applied in the final stage can overcome the slowness of SA in its final stage and helps in achieving higher quality solutions.
- The proposed method, which is a point-to-point method, is promising in practice and competitive with some other population-based methods in terms of the solution qualities. Moreover, the latter methods are more expensive than the proposed method.

Appendix A

Unconstrained Test Problems

(AK_n) Ackley Functions

Definition: $AK_n(\mathbf{x}) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{-\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$.
Search space: $-15 \leq x_i \leq 30,\ i = 1, 2, \ldots, n.$
Global minimum: $\mathbf{x}^* = (0, \ldots, 0);\ AK_n(\mathbf{x}^*) = 0.$

(B_m) Bohachevsky Functions

Definitions: $B_1(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7.$
$B_2(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3.$
$B_3(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1 + 4\pi x_2) + 0.3.$
Search space: $-50 \leq x_i \leq 100,\ i = 1, 2.$
Global minimum: $\mathbf{x}^* = (0, 0);\ B_m(\mathbf{x}^*) = 0,\ m = 1, 2, 3.$

(BL) Beale Function

Definition: $BL(\mathbf{x}) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2.$
Search space: $-4.5 \leq x_i \leq 4.5,\ i = 1, 2.$
Global minimum: $\mathbf{x}^* = (3, 0.5);\ BL(\mathbf{x}^*) = 0.$

(BO) Booth Function

Definition: $BO(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2,$
Search space: $-10 \leq x_i \leq 10,\ i = 1, 2$
Global minimum: $\mathbf{x}^* = (1, 3);\ BO(\mathbf{x}^*) = 0.$

(*CV*) Colville Function

Definition: $CV(\mathbf{x}) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1\left((x_2 - 1)^2 + (x_4 - 1)^2\right) + 19.8(x_2 - 1)(x_4 - 1)$.
Search space: $-10 \leq x_i \leq 10,\ i = 1, \ldots, 4$.
Global minimum: $\mathbf{x}^* = (1, 1, 1, 1);\ CV(\mathbf{x}^*) = 0$.

(*DJ*) De Joung Function

Definition: $DJ(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2$.
Search space: $-2.56 \leq x_i \leq 5.12,\ i = 1, 2, 3$.
Global minimum: $\mathbf{x}^* = (0, 0, 0);\ DJ(\mathbf{x}^*) = 0$.

(DP_n) Dixon&Price Functions

Definition: $DP_n(\mathbf{x}) = (x_1 - 1)^2 + \sum_{i=2}^{n} i\left(2x_i^2 - x_{i-1}\right)^2$.
Search space: $-10 \leq x_i \leq 10,\ i = 1, \ldots, n$.
Global minimum: $x_i^* = 2^{-\left(\frac{2^i - 2}{2^i}\right)},\ i = 1, \ldots, n;\ DP_n(\mathbf{x}^*) = 0$.

(*DX*) Dixon Functions

Definition: $DX(\mathbf{x}) = (1 - x_1)^2 + (1 - x_{10})^2 + \sum_{j=1}^{9}(x_i^2 - x_{i+1})^2$.
Search space: $-10 \leq x_i \leq 10,\ i = 1, \ldots, 10$.
Global minimum: $\mathbf{x}^* = (1, 1, 1, 1);\ DX(\mathbf{x}^*) = 0$.

(*ES*) Easom Function

Definition: $ES(\mathbf{x}) = -\cos(x_1)\cos(x_2)\exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$.
Search space: $-100 \leq x_i \leq 100,\ i = 1, 2$.
Global minimum: $\mathbf{x}^* = (\pi, \pi);\ ES(\mathbf{x}^*) = -1$.

(F_1) Function

Definition: $F_1(x_1, x_2) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2)$.
Search space: $-1 < x_j < 1,\ j = 1, 2$.
Global minimum: $(x_1, x_2)^* = (0, 0);\ F_1((x_1, x_2)^*) = -2$.

(F_2) Function

Definition: $F_2(\mathbf{x}) = \sum_{j=1}^{10} \min\left\{|x_j - 0.2| + a, |x_j - 0.4|, |x_j - 0.7| + a\right\}, a = 0.05.$
Search space: $0 < x_j < 1,\ j = 1, \ldots, 10.$
Global minimum: $\mathbf{x}^* = (0.4, 0.4, \ldots, 0.4);\ F_2(\mathbf{x}^*) = 0.$

($F5$) De Joung Function

Definition: $F5(x_1, x_2) = \left(0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right)^{-1},$
$a_{1j} = -32, -16, 0, 16, 32$ for $j = 1, 2, \ldots, 5,$
$a_{1k} = a_{1j}$ for $k = j + 5, j + 10, j + 15, j + 20,$ and $j = 1, 2, \ldots, 5,$
$a_{2j} = -32, -16, 0, 16, 32$ for $j = 1, 6, 11, 16, 21,$
$a_{2k} = a_{2j}$ for $k = j + 1, j + 2, j + 3, j + 4,$ and $j = 1, 6, 11, 16, 21.$
Search space: $-65.536 < x_i < 65.536,\ i = 1, 2.$
Global minimum: $(x_1, x_2)^* = (-32, -32);\ F5((x_1, x_2)^*) = 0.998004.$

($\boldsymbol{G}_n$) Griewank Functions

Definition: $G_n(\mathbf{x}) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(x_i/\sqrt{i}\right) + 1.$
Search space: $-300 \leq x_i \leq 600,\ i = 1, \ldots, n.$
Global minimum: $\mathbf{x}^* = (0, \ldots, 0),\ G_n(\mathbf{x}^*) = 0.$

($\boldsymbol{GP}$) Goldstein&Price Function

Definition: $GP(\mathbf{x}) = \left(1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 13x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right) *$
$\left(30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 - 48x_2 - 36x_1x_2 + 27x_2^2)\right).$
Search space: $-2 \leq x_i \leq 2,\ i = 1, 2.$
Global minimum: $\mathbf{x}^* = (0, -1);\ GP(\mathbf{x}^*) = 3.$

($\boldsymbol{H}_{3,4}$) Hartmann Function

Definition: $H_{3,4}(\mathbf{x}) = -\sum_{i=1}^{4} \alpha_i \exp\left[-\sum_{j=1}^{3} A_{ij} (x_j - P_{ij})^2\right],$
$\alpha = [1, 1.2, 3, 3.2]^T,\ A = \begin{bmatrix} 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix},\ P = 10^{-4} \begin{bmatrix} 6890 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8828 \end{bmatrix}.$
Search space: $0 \leq x_i \leq 1,\ i = 1, 2, 3.$
Global minimum: $\mathbf{x}^* = (0.114614, 0.555649, 0.852547);\ H_{3,4}(\mathbf{x}^*) = -3.86278.$

($\boldsymbol{H}_{6,4}$) Hartmann Function

Definition: $H_{6,4}(\mathbf{x}) = -\sum_{i=1}^{4} \alpha_i \exp\left[-\sum_{j=1}^{6} B_{ij}\left(x_j - Q_{ij}\right)^2\right]$,

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

$Q = 10^{-4} \begin{bmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{bmatrix}$.

Search space: $0 \le x_i \le 1,\ i = 1, \ldots, 6.$

Global minimum: $\mathbf{x}^* = (0.201690, 0.150011, 0.476874, 0.275332, 0.311652, 0.657300)$; $H_{6,4}(\mathbf{x}^*) = -3.32237.$

(*HM*) Hump Function

Definition: $HM(\mathbf{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4.$

Search space: $-5 \le x_i \le 5,\ i = 1, 2.$

Global minimum: $\mathbf{x}^* = (0.0898, -0.7126),\ (-0.0898, 0.7126);\ HM(\mathbf{x}^*) = 0.$

($\boldsymbol{L}_n$) Levy Functions

Definition: $L_n(\mathbf{x}) = \sin^2(\pi y_1) + \sum_{i=1}^{n-1}\left[(y_i - 1)^2\left(1 + 10\sin^2(\pi y_i + 1)\right)\right]$
$+ (y_n - 1)^2\left(1 + 10\sin^2(2\pi y_n)\right),\ y_i = 1 + \frac{x_i - 1}{4},\ i = 1, \ldots, n.$

Search space: $-10 \le x_i \le 10,\ i = 1, \ldots, n.$

Global minimum: $\mathbf{x}^* = (1, \ldots, 1);\ L_n(\mathbf{x}^*) = 0.$

(*MT*) Matyas Function

Definition: $MT(\mathbf{x}) = 0.26\,(x_1^2 + x_2^2) - 0.48x_1x_2.$

Search space: $-5 \le x_i \le 10,\ i = 1, 2.$

Global minimum: $\mathbf{x}^* = (0, 0);\ MT(\mathbf{x}^*) = 0.$

(MZ) Michalewicz Function

Definition: $MZ(x_1, x_2) = -\sum_{j=1}^{2}\sin(x_j)\left(\sin\left(jx_j^2/\pi\right)\right)^{2m};\ m = 10.$

Search space: $0 \le x_j \le \pi,\ j = 1, 2.$

Global minima: $MZ((x_1, x_2)^*) = -1.8013.$

($\boldsymbol{P}_{n,\beta}$) Perm Functions

Definition: $P_{n,\beta}(\mathbf{x}) = \sum_{k=1}^{n} \left[\sum_{i=1}^{n} \left(i^k + \beta \right) \left((x_i/i)^k - 1 \right) \right]^2$.
Search space: $-n \leq x_i \leq n,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (1, 2, \ldots, n);\ P_{n,\beta}(\mathbf{x}^*) = 0$.

($\boldsymbol{P}^0_{n,\beta}$) Perm Functions

Definition: $P^0_{n,\beta}(\mathbf{x}) = \sum_{k=1}^{n} \left[\sum_{i=1}^{n} \left(i + \beta \right) \left(x_i^k - (1/i)^k \right) \right]^2$.
Search space: $-n \leq x_i \leq n,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (1, \frac{1}{2}, \ldots, \frac{1}{n});\ P^0_{n,\beta}(\mathbf{x}^*) = 0$.

($\boldsymbol{PS}_{b_1,\ldots,b_n}$) Power Sum Functions

Definition: $PS_{b_1,\ldots,b_n}(\mathbf{x}) = \sum_{k=1}^{n} \left[\left(\sum_{i=1}^{n} x_i^k \right) - b_k \right]^2$.
Search space: $0 \leq x_i \leq n,\ i = 1, \ldots, n$.
Global minimum for $\boldsymbol{PS}_{8,18,44,114}(\mathbf{x})$: $\mathbf{x}^* = (1, 2, 2, 3);\ PS_{8,18,44,114}(\mathbf{x}^*) = 0$.

($\boldsymbol{PW}_n$) Powell Functions

Definition: $PW_n(\mathbf{x}) = \sum_{i=1}^{n/4} \left(x_{4i-3} + 10x_{4i-2} \right)^2 + 5 \left(x_{4i-1} - x_{4i} \right)^2 + \left(x_{4i-2} - x_{4i-1} \right)^4 + 10 \left(x_{4i-3} - x_{4i} \right)^4$.
Search space: $-4 \leq x_i \leq 5,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (3, -1, 0, 1, 3, \ldots, 3, -1, 0, 1);\ PW_n(\mathbf{x}^*) = 0$.

($\boldsymbol{R}_n$) Rosenbrock Functions

Definition: $R_n(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100 \left(x_i^2 - x_{i+1} \right)^2 + \left(x_i - 1 \right)^2 \right]$.
Search space: $-5 \leq x_i \leq 10,\ i = 1, 2, \ldots, n$.
Global minimum: $\mathbf{x}^* = (1, \ldots, 1),\ R_n(\mathbf{x}^*) = 0$.

($\boldsymbol{RC}$) Branin RCOS Function

Definition: $RC(\mathbf{x}) = (x_2 - \frac{5}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6)^2 + 10(1 - \frac{1}{8\pi}) \cos(x_1) + 10$.
Search space: $-5 \leq x_1 \leq 10,\ 0 \leq x_2 \leq 15$.
Global minima: $\mathbf{x}^* = (-\pi, 12.275), (\pi, 2.275), (9.42478, 2.475);\ RC(\mathbf{x}^*) = 0.397887$.

(RT_n) Rastrigin Functions

Definition: $RT_n(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos\left(2\pi x_i\right)\right)$.
Search space: $-2.56 \leq x_i \leq 5.12,\ i = 1, \ldots, n.$
Global minimum: $\mathbf{x}^* = (0, \ldots, 0),\ RT_n(\mathbf{x}^*) = 0.$

($S_{4,m}$) Shekel Functions

Definition: $S_{4,m}(\mathbf{x}) = -\sum_{j=1}^{m} \left[\sum_{i=1}^{4} \left(x_i - C_{ij}\right)^2 + \beta_j\right]^{-1}$,
$\beta = \frac{1}{10}\left[1, 2, 2, 4, 4, 6, 3, 7, 5, 5\right]^T$, $C = \begin{bmatrix} 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 5.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 5.0 & 1.0 & 2.0 & 3.6 \\ 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 3.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 3.0 & 1.0 & 2.0 & 3.6 \end{bmatrix}$.
Search space: $0 \leq x_i \leq 10,\ i = 1, \ldots, 4.$
Global minima: $\mathbf{x}^* = (4, 4, 4, 4)$; $S_{4,5}(\mathbf{x}^*) = -10.1532$, $S_{4,7}(\mathbf{x}^*) = -10.4029$ and $S_{4,10}(\mathbf{x}^*) = -10.5364$.

(SC_n) Schwefel Functions

Definition: $SC_n(\mathbf{x}) = 418.9829n - \sum_{i=1}^{n} \left(x_i \sin\sqrt{|x_i|}\right)$.
Search space: $-500 \leq x_i \leq 500,\ i = 1, 2, \ldots, n.$
Global minimum: $\mathbf{x}^* = (1, \ldots, 1),\ SC_n(\mathbf{x}^*) = 0.$

(SH) Shubert Function

Definition: $SH(\mathbf{x}) = \left(\sum_{i=1}^{5} i\cos\left(\left(i+1\right)x_1 + i\right)\right)\left(\sum_{i=1}^{5} i\cos\left(\left(i+1\right)x_2 + i\right)\right)$,
Search space: $-10 \leq x_i \leq 10,\ i = 1, 2$
Global minima: 18 global minima and $SH(\mathbf{x}^*) = -186.7309$.

(SR_n) Sphere Functions

Definition: $SR_n(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$,
Search space: $-2.56 \leq x_i \leq 5.12,\ i = 1, \ldots, n$
Global minimum: $\mathbf{x}^* = (0, \ldots, 0),\ SR_n(\mathbf{x}^*) = 0.$

(SS_n) Sum Squares Functions

Definition: $SS_n(\mathbf{x}) = \sum_{i=1}^{n} i x_i^2$,
Search space: $-5 \leq x_i \leq 10,\ i = 1, \ldots, n$
Global minimum: $\mathbf{x}^* = (0, \ldots, 0),\ SS_n(\mathbf{x}^*) = 0.$

(T_n) Trid Functions

Definition: $T_n(\mathbf{x}) = \sum_{i=1}^{n} (x_i - 1)^2 - \sum_{i=2}^{n} x_i x_{i-1}$,
Search space: $-n^2 \leq x_i \leq n^2,\ i = 1, \ldots, n$
Global minima: *a)* $n = 6,\ x_i^* = i\,(7 - i)\,,\ i = 1, \ldots, n,\ T_n(\mathbf{x}^*) = -50$,
b) $n = 10,\ x_i^* = i\,(11 - i)\,,\ i = 1, \ldots, n,\ T_n(\mathbf{x}^*) = -210$,

(Z_n) Zakharov Functions

Definition: $Z_n(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{n} 0.5 i x_i\right)^4$.
Search space: $-5 \leq x_i \leq 10,\ i = 1, 2, \ldots, n$
Global minimum: $\mathbf{x}^* = (0, \ldots, 0)$, $Z_n(\mathbf{x}^*) = 0$.

Appendix B

Constrained Test Problems

Problem G1[1]

$$\begin{aligned}
\min_x\ & f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i, \\
\text{s.t.}\ & g_1(x) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \le 0, \\
& g_2(x) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \le 0, \\
& g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \le 0, \\
& g_4(x) = -8x_1 + x_{10} \le 0, \\
& g_5(x) = -8x_2 + x_{11} \le 0, \\
& g_6(x) = -8x_3 + x_{12} \le 0, \\
& g_7(x) = -2x_4 - x_5 + x_{10} \le 0, \\
& g_8(x) = -2x_6 - x_7 + x_{11} \le 0, \\
& g_9(x) = -2x_8 - x_9 + x_{12} \le 0, \\
& x_i \ge 0,\ i = 1, \ldots, 13, \\
& x_i \le 1,\ i = 1, \ldots, 9, 13.
\end{aligned}$$

The bounds: $U = (1, 1, 1, 1, 1, 1, 1, 1, 1, 100, 100, 100, 1)$ and $L = (0, \ldots, 0)$.
Global minimum: $x^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$, $f(x^*) = -15$.

Problem G2

$$\begin{aligned}
\max_x\ & f(x) = \left|\frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}}\right|, \\
\text{s.t.}\ & g_1(x) = -\prod_{i=1}^{n} x_i + 0.75 \le 0, \\
& g_2(x) = \sum_{i=1}^{n} x_i - 7.5n \le 0.
\end{aligned}$$

The bounds: $U = (10, \ldots, 10)$ and $L = (0, \ldots, 0)$.
Best known: $f(x^*) = 0.803619$, for $n = 20$.

[1]The formula of G1 is presented as its common form in the literature [25]. However, variable x_{13} can be eliminated since its value of the global solution, which is $x_{13} = 1$, can be easily derived.

Problem G3

$$\begin{aligned} \max_x \; & f(x) = (\sqrt{n})^n \prod_{i=1}^n x_i, \\ \text{s.t.} \; & h_1(x) = \sum_{i=1}^n x_i^2 - 1 = 0 \end{aligned}$$

The bounds: $U = (1, \ldots, 1)$ and $L = (0, \ldots, 0)$.
Global minimum: $x^* = \left(\frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}}\right)$, $f(x^*) = 1$.

Problem G4

$$\begin{aligned} \min_x \; & f(x) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141, \\ \text{s.t.} \; & g_1(x) = u(x) - 92 \le 0, \\ & g_2(x) = -u(x) \le 0, \\ & g_3(x) = v(x) - 110 \le 0, \\ & g_4(x) = -v(x) + 90 \le 0, \\ & g_5(x) = w(x) - 25 \le 0, \\ & g_6(x) = -w(x) + 20 \le 0, \end{aligned}$$

where

$$\begin{aligned} u(x) &= 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5, \\ v(x) &= 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2, \\ w(x) &= 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4. \end{aligned}$$

The bounds: $U = (102, 45, 45, 45, 45)$ and $L = (78, 33, 27, 27, 27)$.
Global minimum: $x^* = (78, 33, 29.995256025682, 45, 36.775812905788)$,
$f(x^*) = -30665.539$.

Problem G5

$$\begin{aligned} \min_x \; & f(x) = 3x_1 + 10^{-6}x_1^3 + 2x_2 + \tfrac{2}{3} \times 10^{-6}x_2^3, \\ \text{s.t.} \; & g_1(x) = x_3 - x_4 - 0.55 \le 0, \\ & g_2(x) = x_4 - x_3 - 0.55 \le 0, \\ & h_1(x) = 1000\left[\sin(-x_3 - 0.25) + \sin(-x_4 - 0.25)\right] + 894.8 - x_1 = 0, \\ & h_2(x) = 1000\left[\sin(x_3 - 0.25) + \sin(x_3 - x_4 - 0.25)\right] + 894.8 - x_2 = 0, \\ & h_3(x) = 1000\left[\sin(x_4 - 0.25) + \sin(x_4 - x_3 - 0.25)\right] + 1294.8 = 0. \end{aligned}$$

The bounds: $U = (1200, 1200, 0.55, 0.55)$ and $L = (0, 0, -0.55, -0.55)$.
Best known: $x^* = (679.9453, 1026, 0.118876, -0.3962336)$, $f(x^*) = 5126.4981$.

Problem G6

$$\begin{aligned}\min_x\ & f(x) = (x_1-10)^3 + (x_2-20)^3,\\ \text{s.t.}\ & g_1(x) = (x_1-5)^2 + (x_2-5)^2 + 100 \le 0,\\ & g_2(x) = (x_1-5)^2 + (x_2-5)^2 - 82.81 \le 0.\end{aligned}$$

The bounds: $U = (100, 100)$ and $L = (13, 0)$.
Global minimum: $x^* = (14.095, 0.84296)$, $f(x^*) = -6961.81388$.

Problem G7

$$\begin{aligned}\min_x\ & f(x) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3-10)^2 + 4(x_4-5)^2 + (x_5-3)^2\\ & \quad +2(x_6-1)^2 + 5x_7^2 + 7(x_8-11)^2 + 2(x_9-10)^2 + (x_{10}-7)^2 + 45,\\ \text{s.t.}\ & g_1(x) = 4x_1 + 5x_2 - 3x_7 + 9x_8 - 105 \le 0,\\ & g_2(x) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \le 0,\\ & g_3(x) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \le 0,\\ & g_4(x) = 3(x_1-2)^2 + 4(x_2-3)^2 + 2x_3^2 - 7x_4 - 120 \le 0,\\ & g_5(x) = 5x_1^2 + 8x_2 + (x_3-6)^2 - 2x_4 - 40 \le 0,\\ & g_6(x) = 0.5(x_1-8)^2 + 2(x_2-4)^2 + 3x_5^2 - x_6 - 30 \le 0,\\ & g_7(x) = x_1^2 + 2(x_2-2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \le 0,\\ & g_8(x) = -3x_1 + 6x_2 + 12(x_9-8)^2 - 7x_{10} \le 0.\end{aligned}$$

The bounds: $U = (10, \ldots, 10)$ and $L = (-10, \ldots, -10)$.
Global minimum: $x^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927)$, $f(x^*) = 24.3062091$.

Problem G8

$$\begin{aligned}\max_x\ & f(x) = \tfrac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1+x_2)},\\ \text{s.t.}\ & g_1(x) = x_1^2 - x_2 + 1 \le 0,\\ & g_2(x) = 1 - x_1 + (x_2-4)^2 \le 0.\end{aligned}$$

The bounds: $U = (10, 10)$ and $L = (0, 0)$.
Global minimum: $x^* = (1.2279713, 4.2453733)$, $f(x^*) = 0.095825$.

Problem G9

$$\begin{aligned}
\min_x \; & f(x) = (x_1-10)^2 + 5(x_2-12)^2 + x_3^4 + 3(x_4-11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 \\
& \qquad -10x_6 - 8x_7, \\
\text{s.t.} \; & g_1(x) = 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 - 127 \le 0, \\
& g_2(x) = 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 - 282 \le 0, \\
& g_3(x) = 23x_1 + x_2^2 + 6x_6^2 - 8x_7 - 196 \le 0, \\
& g_4(x) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \le 0.
\end{aligned}$$

The bounds: $U = (10, \ldots, 10)$ and $L = (-10, \ldots, -10)$.
Global minimum: $x^* = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$, $f(x^*) = 680.6300573$.

Problem G10

$$\begin{aligned}
\min_x \; & f(x) = x_1 + x_2 + x_3, \\
\text{s.t.} \; & g_1(x) = -1 + 0.0025(x_4 + x_6) \le 0, \\
& g_2(x) = -1 + 0.0025(-x_4 + x_5 + x_7) \le 0, \\
& g_3(x) = -1 + 0.01(-x_5 + x_8) \le 0, \\
& g_4(x) = 100x_1 - x_1x_6 + 833.33252x_4 - 83333.333 \le 0, \\
& g_5(x) = x_2x_4 - x_2x_7 - 1250x_4 + 1250x_5 \le 0, \\
& g_6(x) = x_3x_5 - x_3x_8 - 2500x_5 + 1250000 \le 0.
\end{aligned}$$

The bounds: $U = (10000, 10000, 10000, 1000, 1000, 1000, 1000, 1000)$ and $L = (100, 1000, 1000, 10, 10, 10, 10, 10)$.
Global minimum: $x^* = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)$, $f(x^*) = 7049.3307$.

Problem G11

$$\begin{aligned}
\min_x \; & f(x) = x_1^2 + (x_2 - 1)^2, \\
\text{s.t.} \; & h_1(x) = x_2 - x_1^2 = 0.
\end{aligned}$$

The bounds: $U = (1, 1)$ and $L = (-1, -1)$.
Global minimum: $x^* = \left(\pm\frac{1}{\sqrt{2}}, \frac{1}{2}\right)$, $f(x^*) = 0.75$.

Problem G12

$$\begin{aligned} \min_{x} \; & f(x) = 1 - 0.01[(x_1 - 5)^2 + (x_2 - 5)^2 + (x_3 - 5)^2], \\ \text{s.t.} \; & g_{i,j,k}(x) = (x_1 - i)^2 + (x_2 - j)^2 + (x_3 - k)^2 - 0.0625 \leq 0, \end{aligned}$$

where $i, j, k = 1, 2, \ldots, 9$.

The bounds: $U = (10, 10, 10)$ and $L = (0, 0, 0)$.

Global minimum: $x^* = (5, 5, 5)$, $f(x^*) = 1$.

Problem G13

$$\begin{aligned} \min_{x} \; & f(x) = e^{x_1 x_2 x_3 x_4 x_5}, \\ \text{s.t.} \; & h_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0, \\ & h_2(x) = x_2 x_3 - 5 x_4 x_5 = 0, \\ & h_3(x) = x_1^3 + x_2^3 + 1 = 0. \end{aligned}$$

The bounds: $U = (2.3, 2.3, 3.2, 3.2, 3.2)$ and $L = (-2.3, -2.3, -3.2, -3.2, -3.2)$.

Global minimum: $x^* = (-1.717143, 1.595709, 1.827247, -0.7636413, -0.763645)$, $f(x^*) = 0.0539498$.

Bibliography

[1] E. Aarts and J. Korst, Selected topics in simulated annealing, in: C.C. Ribeiro and P. Hansen (Eds.), *Essays and Surveys in Metaheuristics*, Kluwer Academic Publishers, Boston, MA, (2002) 1–37.

[2] M. A. Abramson, C. Audet, and J. E. Dennis Jr., Generalized pattern searches with derivative information, *Mathematical Programming*, 100 (2004) 3-25.

[3] K.S. Al-Sultan and M. A. Al-Fawzan, A tabu search Hooke and Jeeves algorithm for unconstrained optimization, *European J. of Operational Research*, 103 (1997) 198–208.

[4] C. Audet and J. E. Dennis, Jr., Analysis of generalized pattern searches, *SIAM Journal on Optimization*, 13(3) (2003) 889–903.

[5] C. Audet and J.E. Dennis Jr., A pattern search filter method for nonlinear programming without derivatives, *SIAM Journal on Optimization*, to appear.

[6] T. Bäck and F. Hoffmeister, Extended selection mechanisms in genetic algorithms, in: R. Belew and L.B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmmann, San Mateo, (1991) 92–99.

[7] T. Bäck, D.B. Fogel and Z. Michalewicz (Eds.), Evolutionary Computation 1: Basic Algorithms and Operators, Institute of Physics Publishing, 2000.

[8] T. Bäck, D.B. Fogel and Z. Michalewicz (Eds.), Evolutionary Computation 2: Advanced Algorithms and Operators, Institute of Physics Publishing, 2000.

[9] J. E. Baker, Adaptive selection methods for genetic algorithms. In: J. J. Grefenstette (Ed.), *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, MA, (1985) 101–111.

[10] R. Battiti and G. Tecchiolli, The continuous reactive tabu search: Blending combinatorial optimization and stochastic search for global optimization, *Annals of Operations Research*, 63 (1996) 153–188.

[11] M. Bessaou and P. Siarry, A genetic algorithm with real-value coding to optimize multimodal continuous functions, *Struct Multidisc Optim*, 23 (2001) 63–74.

[12] I. Bohachevsky, M. E. Johnson and M. L. Stein, Generalized simulated annealing for function optimization, *Technometrics*, 28 (1986) 209–217.

[13] M. F. Cardoso, R. L. Salcedo and S. F. de Azevedo, The simplex-simulated annealing approach to continuous non-linear optimization, *Comput. Chem. Eng.*, 20 (1996) 1065–1080.

[14] M. F. Cardoso, R. L. Salcedo, S. F. de Azevedo and D. Barbosa, A simulated annealing approach to the solution of minlp problems, *Comput. Chem. Eng.*, 21 (1997) 1349–1364.

[15] R. Chelouah and P. Siarry, Tabu search applied to global optimization, *European J. of Operational Reasearch*, 123 (2000) 256–270.

[16] R. Chelouah and P. Siarry, A continuous genetic algorithm designed for the global optimization of multimodal functions, *J. Heuristics*, 6 (2000) 191–213.

[17] Y.X. Chen, Optimal Anytime Search for Constrained Nonlinear Programming, M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois, May 2001.

[18] C. A. Coello Coello, Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art, *Computer Methods in Applied Mechanics and Engineering*, 191 (2002), 1245–1287.

[19] C. A. Coello Coello and E. M. Montes, Constraint-handling in genetic algorithms through the use of dominance-based tournament selection, *Advanced Engineering Informatics*, 16 (2002), 193–203.

[20] D. Cvijovic and J. Klinowski, Taboo search: An approach to the multiple minima problem, *Science*, 667 (1995) 664–666.

[21] D. Cvijovic and J. Klinowski, Taboo search: An approach to the multiple-minima problem for continuous functions, in: P. M. Pardalos and H. E. Romeijn (Eds.), *Handbook of Global Optimization*, Kluwer Academic Publishers, Boston, MA, (2002) 387–406.

[22] K. Deb, An efficient constraint handling method for genetic algorithms, *Computer Methods in Applied Mechanics and Engineering*, 186 (2000), 311–338.

[23] J. E. Dennis and V. Torczon, Direct search methods on parallel machines, *SIAM J. Optim.*, 1 (1991) 448–474.

[24] R. Fletcher and S. Leyffer, Nonlinear programming without a penalty function, *Mathematical Programming*, 91 (2002), 239–269.

[25] C. A. Floudas, P. M. Pardalos, C. S. Adjiman, W. R. Esposito, Z. Gumus, S. T. Harding, J. L. Klepeis, C. A. Meyer and C. A. Schweiger (Eds.), Handbook of Test Problems for Local and Global Optimization, Kluwer Academic Publishers, Boston, MA, 1999.

[26] F. Franze and N. Speciale, A tabu-search-based algorithm for continuous multiminima problems, *International Journal for Numerical Engineering*, 50 (2001) 665–680.

[27] F. Glover, Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research*, 13(5) (1986) 533–549.

[28] F. Glover, Tabu search–Part I, *ORSA Journal on Computing*, 1 (1989) 190–206.

[29] F. Glover, Tabu search–Part II, *ORSA Journal on Computing*, 2 (1990) 4–32.

[30] F. Glover and G. Kochenberger (Eds.), Handbook of MetaHeuristics, Kluwer Academic Publishers, Boston, MA, 2002.

[31] F. Glover and M. Laguna, Tabu Search, Kluwer Academic Publishers, MA, USA, 1997.

[32] F. Glover and M. Laguna, Tabu search, in: P. M. Pardalos and M. G. C. Resende (Eds.), *Handbook of Applied Optimization*, Oxford University Press, (2002) 194–208.

[33] F. Glover, E. Taillard and D. Werra, A user's guide to tabu search, *Annals of Operations Research*, 41 (1993) 3–28.

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

[35] T. Günal, A hybrid approach to the synthesis of nonuniform lossy transmission-line impedance-matching sections, *Microwave and Optical Technology Letters*, 24 (2000) 121–125.

[36] S. B. Hamida and M. Schoenauer. ASCHEA: New rsults using adaptive segregational constraint handling, in: *Proceedings of the Congress on Evolutionary Computation (CEC2002)*, Piscataway, New Jersey, IEEE Service Center, 2002, pp. 884–889.

[37] A. Hedar and M. Fukushima, Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization, *Optimization Methods and Software*, 17 (2002) 891–912.

[38] A. Hedar and M. Fukushima, Minimizing multimodal functions by simplex coding genetic algorithm, *Optimization Methods and Software*, 18 (2003) 265–282.

[39] A. Hedar and M. Fukushima, Tabu Search directed by direct search methods for nonlinear global optimization, *European J. of Operational Reasearch*, to appear.

[40] A. Hedar and M. Fukushima, Simplex coding genetic algorithm for the global optimization of nonlinear functions, in: T. Tanino, T. Tanaka and M. Inuiguchi (Eds.), *Multi-Objective Programming and Goal Programming*, Springer-Verlag, Berlin-Heidelberg, 2003, pp. 135–140.

[41] A. Hedar and M. Fukushima, Heuristic pattern search and its hybridization with simulated annealing for nonlinear global optimization, *Optimization Methods and Software*, 19 (2004) 291–308.

[42] A. Hedar and M. Fukushima, Derivative-free filter simulated annealing method for constrained continuous global optimization, Technical Report 2004-007, Department of Applied Mathematics and Physics, Kyoto University, April (2004).

[43] D. Henderson, S. H. Jacobson and A. W. Johnson, The thoery and practice of simulated annealing, in: F. Glover and G. Kochenberger (Eds.), *Handbook of MetaHeuristics*, Kluwer Academic Publishers, Boston, MA, (2002) 287–319.

[44] F. Herrera, M. Lozano and J.L. Verdegay, Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis, Artificial Intelligence Review, 12 (1998) 265-319.

[45] W. Hock and K. Schittkowski, Test Examples for Nonlinear Programming Codes, Springer-Verlag Berlin Heidelberg, 1981.

[46] J. Holland, Adaptation in Natural and Artifical Systems, University of Michigan Press, Ann Arbor, 1975.

[47] R. Hooke and T. A. Jeeves, Direct search solution of numerical and statistical problems, *J. ACM*, 8 (1961) 212-229.

[48] R. Horst and P. M. Pardalos (Eds), Handbook of Global Optimization, Kluwer Academic Publishers, Boston, MA, 1995.

[49] R. Horst, N.V. Thoai and P.M. Pardalos, Introduction to Global Optimization, Second Edition, Kluwer Academic Publishers, Boston, MA, 2000.

[50] N. Hu, Tabu search method with random moves for globally optimal design, *International Journal for Numerical Engineering*, 35 (1992) 1055–1070.

[51] C. T. Kelley, Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition, *SIAM J. Optim.*, 10 (1999) 43–55.

[52] C. T. Kelley, Iterative Methods for Optimization, Frontiers Appl. Math. 18, SIAM, Philadelphia, PA, 1999.

[53] S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi, Optimisation by simulated annealing, *Science*, 220 (1983) 671–680.

[54] T.G. Kolda, R.M. Lewis and V. Torczon, Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45 (2003) 385–482.

[55] S. Koziel and Z. Michalewicz, Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization, *Evolutionary Computation*, 7(1) (1999), 19–44.

[56] V. Kvasnicka and J. Pospichal, A hybrid of simplex method and simulated annealing, *Chemometrics and Intelligent Laboratory Systems*, 39 (1997) 161-173.

[57] P. J. Laarhoven, Theoretical and Computational Aspects of Simulated Annealing, Stichting Mathematisch Centrum, Amsterdam, 1988.

[58] P. J. Laarhoven and E. H. Aarts, Simulated Annealing: Theory and Applications, D. Reidel Publishing Company, Dordrecht, Holland, 1987.

[59] M. Laguna and R. Martí, Experimental testing of advanced scatter aearch designs for global optimization of multimodal functions, Technical Report, University of Colorado at Boulder, November (2002).

[60] M. Laguna and R. Marti, Scatter Search: Methodology and Implementations in C, Kluwer Academic Publishers, Boston, 2003.

[61] M. Laguna, R. Martí and V. Campos, Intensification and diversification with elite tabu search solutions for the linear ordering problem, *Computers and Operations Research* 26 (1999) 1217–1230.

[62] R. Martí, Multi-Start Methods , in: F. Glover and G. Kochenberger (Eds.), Handbook of MetaHeuristics, Kluwer Academic Publishers, Boston, MA, (2002) 355–368.

[63] R. Martí and J.M. Moreno, Métodos multi-arranque, Inteligencia Artificial 19 (2003) 49–60.

[64] K.I.M. McKinnon, Convergence of the Nelder-Mead simplex method to a nonstationary point, *SIAM J. Optim.*, 9 (1999) 148–158.

[65] M. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, Equation of state calculation by fast computing machines. Journal of Chemical Physics, 21 (1953) 1087–1092.

[66] E. Mezura-Montes and C. A. Coello Coello. A simple multimembered evolution strategy to solve constrained optimization problems, Technical Report EVOCINV-04-2003, Evolutionary Computation Group at CINVESTAV, Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, México D.F., México, 2003.

[67] Z. Michalewicz and G. Nazhiyath, Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints, *Proceedings of the 2nd IEEE International Conference on Evolutionary Computation*, Preth, Australia, (1995) 647–651.

[68] Z. Michalewicz, Genetic algorithms + Data structures = Evolution programs, Springer, Berlin, Heidelberg, New York, 1996.

[69] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems, Evolutionary Computation, 4(1) (1996), 1–32.

[70] P. Moscato, Memetic algorithms: An introduction, in: D. Corne, M. Dorigo and F. Glover (Eds.), *New Ideas in Optimization*. McGraw-Hill, London, UK, (1999).

[71] M. Musil, M. J. Wilmut and R. Chapman, A hybrid simplex genetic algorithm for estimating geoacoustic parameters using matched-field inversion, *IEEE J. Oceanic Eng.*, 24 (1999) 358–369.

[72] J. A. Nelder and R. Mead, A simplex method for function minimization, *Comput. J.*, 7 (1965) 308-313.

[73] I. H. Osman and J. P. Kelly, Meta-Heuristics: Theory and Applications, Kluwer Academic Publishers, Boston, MA, 1996.

[74] P. M. Pardalos and M. G. C. Resende (Eds.), Handbook of Applied Optimization, Oxford University Press, Oxford, 2002.

[75] P.M. Pardalos and H. E. Romeijn (Eds.), Handbook of Global Optimization, Kluwer Academic Publishers, Boston, MA, 2002.

[76] P. M. Pardalos, H. E. Romeijn and H. Tuy, Recent developments and trends in global optimization, *J. Comput. Appl. Math.*, 124 (2000) 209–228.

[77] M.J.D. Powell, A direct search optimization method that models the objective and constraint functions by linear interpolation, in: S. Gomez and J.-P. Hennart (Eds.), *Advances in Optimization and Numerical Analysis*, Math. Appl. 275, Kluwer Academic, Dordrecht, The Netherlands, (1994) 51-67.

[78] M.J.D. Powell, Direct search algorithms for optimization calculations, *Acta Numer.*, 7 (1998) 287-336.

[79] W. H. Press and S. A. Teukolsky, Simulated annealing optimization over continuous spaces, *Comput. Phys.*, 5 (1991) 426–429.

[80] J. M. Renders and H. Bersini, Hybridizing genetic algorithms with hill-climbing methods for global optimization: Two possible ways, in: Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel. D. B. Fogel, and H. Kitano (Eds.), *Proceedings of the First IEEE Conference on Evolutionary Computation*, IEEE Press, (1994) 312–317.

[81] C.C. Ribeiro and P. Hansen (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, Boston, MA, 2002.

[82] H. E. Romeijn and R. L. Smith, Simulated annealing for global constrained optimization, *Journal Of Global Optimization*, 5 (1994), 101–126.

[83] T. P. Runarsson and X. Yao, Stochastic Ranking for Constrained Evolutionary Optimization. *IEEE Transactions on Evolutionary Computation*, 4(3) (2000), 284–294.

[84] F. Schoen, Two phase methods for globale optimization, in: P. M. Pardalos and H. E. Romeijn (Eds.), *Handbook of Global Optimization*, Kluwer Academic Publishers, Boston, MA, (2002) 151-178.

[85] P. Siarry, G. Berthiau, F. Durbin and J. Haussy, Enhanced simulated annealing for globally minimizing functions of many continuous variables, *ACM Transactions on Mathematical Software*, 23 (1997) 209–228.

[86] P. Tseng, Fortified-descent simplicial search method: A general approach, *SIAM J. Optim.*, 10 (1999) 269–288.

[87] V. Torczon, Multi-Directional Search: A Direct Search Algorithm for Parallel Machines, Ph.D. thesis, Department of Mathematical Sciences, Rice University, Houston, TX, 1989.

[88] V. Torczon, On the convergence of pattern search algorithms. *SIAM J. Optim.*, 7 (1997) 1–25.

[89] B. W. Wah and Y. X. Chen, Optimal anytime constrained simulated annealing for constrained global optimization, in: R. Dechter (Ed.), LNCS 1894, Springer-Verlag, Sept. 2000, pp. 425–440.

[90] B. W. Wah and T. Wang, Tuning strategies of constrained simulated annealing for nonlinear global optimization, *International Journal on Artificial Intelligence Tools*, 9(1) (2000), 3–25.

[91] T. Wang, Global Optimization of Constrained Nonlinear Programming, Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Dec. 2000.

[92] P. P. Wang and D. S. Chen, Continuous optimization by a variant of simulated annealing, *Computational Optimization and Applications*, 6 (1996) 59–71.

[93] M.H. Wright, Direct search methods: Once scorned, now respectable, in: D. F. Griffiths and G. A. Watson, (Eds.), *Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis*, Pitman Res. Notes Math. Ser. 344, CRC Press, Boca Raton, FL, (1996) 191-208.

[94] R. Yang and I. Douglas, Simple genetic algorithm with local tuning: Efficient global optimizing technique, *J. Optim. Theory Appl.*, 98 (1998) 449–465.

[95] J. Yen, J. C. Liao, B. Lee and D. Randolph, A hybrid approach to modeling metabolic systems using a genetic algorithm and simplex method, *IEEE Trans. on Syst., Man, and Cybern. B*, 28 (1998) 173–191.

[96] R. Zentner, Z. Sipus and J. Bartolic, Optimization synthesis of broadband circularly polarized microstrip antennas by hybrid genetic algorithm, *Microwave and Optical Technology Letters*, 31 (2001) 197–201.