Some Population Set-Based Methods for Unconstrained Global Optimization

Professor Kaelo

A thesis submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Doctor of Philosophy.

Johannesburg, 2005

Declaration

I declare that this thesis is my own, unaided work. It is being submitted for the Degree of Doctor of Philosophy in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

_____

(Signature of candidate)

_____

(Date)

Abstract

Many real-life problems are formulated as global optimization problems with continuous variables. These problems are in most cases nonsmooth, nonconvex and often simulation based, making gradient based methods impossible to be used to solve them. Therefore, efficient, reliable and derivative-free global optimization methods for solving such problems are needed.

In this thesis, we focus on improving the efficiency and reliability of some global optimization methods. In particular, we concentrate on improving some population set-based methods for unconstrained global optimization, mainly through hybridization. Hybridization has widely been recognized to be one of the most attractive areas of unconstrained global optimization. Experiments have shown that through hybridization, new methods that inherit the strength of the original elements but not their weakness can be formed. We suggest a number of new hybridized population set-based methods based on differential evolution (de), controlled random search (crs2) and real coded genetic algorithm (ga).

We propose five new versions of de. In the first version, we introduce a localization, called random localization, in the mutation phase of de. In the second version, we propose a localization in the acceptance phase of de. In the third version, we form a de hybrid algorithm by probabilistically combining the point generation scheme of crs2 with that of de in the de algorithm. The fourth and fifth versions are also de hybrids. These versions hybridize the mutation of de with the point generation rule of the electromagnetism-like (em) algorithm. We also propose five new versions of crs2. The first version modifies the point generation scheme of crs2 by introducing a local mutation technique. In the second and third modifications, we probabilistically combine the point generation scheme of crs2 with the linear interpolation scheme of a trust-region based method. The fourth version is a crs hybrid that probabilistically combines the quadratic interpolation scheme with the linear interpolation scheme in crs2. In the fifth version,

we form a crs2 hybrid algorithm by probabilistically combining the point generation scheme of crs2 with that of de in the crs2 algorithm. Finally, we propose five new versions of the real coded genetic algorithm (ga) with arithmetic crossover. In the first version of ga, we introduce a local technique. We propose, in the second version, an integrated crossover rule that generates two children at a time using two different crossover rules. We introduce a local technique in the second version to obtain the third version. The fourth and fifth versions are based on the probabilistic adaptation of crossover rules.

The efficiency and reliability of the new methods are evaluated through numerical experiments using a large test suite of both simple and difficult problems from the literature. Results indicate that the new hybrids are much better than their original counterparts both in reliability and efficiency. Therefore, the new hybrids proposed in this study offer an alternative to many currently available stochastic algorithms for solving global optimization problems in which the gradient information is not readily available.

Keywords: Global optimization, genetic algorithms, operations research, direct search methods, probabilistic adaptation, differential evolution, controlled random search, nonlinear optimization, heuristics, hybridization, programming model, derivative-free, optimization problem.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Montaz Ali, who has endlessly and tirelessly supervised this thesis. His valuable supervision has significantly improved this thesis. He also co-authored in all the papers. I appreciate his many useful comments on this work and also appreciate his advice and comments that significantly improved the thesis and the papers in both scientific and linguistic aspects. I know of no supervisor as dedicated and committed to his students.

Many thanks to my wife and son, for the love and encouragement they showed throughout my stay in South Africa.

Last but not least, many thanks to my family and friends for always supporting me. Lesh and Peace, thanks guys for all the time we had together, especially at the gym! I know I always came up with excuses whenever I felt like dodging the gym! Lifting those weights was not as easy! Lesh knows better!

Contents

List of Figures

List of Tables

LIST OF INCLUDED ARTICLES

[A] P. Kaelo and M.M. Ali, A numerical study of some modified differential evolution algorithms. European Journal of Operational Research 169 (3), (2006) 1176-1184.

[B] P. Kaelo and M.M. Ali, Some variants of the controlled random search algorithm for global optimization. Submitted to Journal of Optimization Theory and Applications.

[C] P. Kaelo and M.M. Ali, Probabilistic adaptation of point generation schemes in some global optimization algorithms. Optimization Methods and Software, in press.

[D] P. Kaelo and M.M. Ali, Differential evolution algorithms using hybrid mutation. Submitted to Computational Optimization and Applications.

[E] P. Kaelo and M.M. Ali, Integrated crossover rules in real coded genetic algorithms. European Journal of Operational Research, in press.

Chapter 1

Introduction

Global optimization is a very important branch of optimization. Optimization is the process of finding the best possible solution to an optimization problem within (often) a given time limit. An optimization problem, in turn, is a problem for which they are different possible solutions. An optimization problem can be classified as a local optimization problem or a global optimization problem. For a local optimization problem, any possible solution is sufficient while in global optimization, the task is to find the absolutely best solution. Global optimization problems are generally difficult to solve, yet they are prevalent in many practical applications. These applications include, but are not limited to, areas of engineering, operations research and molecular biology [34, 35, 51, 102]. In engineering, for example, optimization problems arise in engineering design and applications. In operations research, decisions are made by developing optimization models that describe the nature of the problem and then mathematical procedures are applied to solve these models while in molecular biology, problems arise as models of molecular structures. Many of these optimization problems may have functions that are provided as 'black-box', that is, the function might be a procedure that gives a function evaluation for a specific point in the search space.

When solving an optimization problem, it is important to consider the mathematical structure of the objective function, the decision variables and the constraints of the problem. The objective function might be continuous or noncontinuous, linear or nonlinear and convex or nonconvex. The decision variables may be discrete or continuous. The feasible search region may be convex or nonconvex. These differences have a great impact on how an optimization

problem can be solved. Also, these differences are used to classify optimization problems. A problem with linear objective function and linear constraints is called a linear programming (optimization) problem. A problem in which either the objective function or any of the constraints is nonlinear is called a nonlinear optimization problem. In the above two cases, the decision variables are assumed to be continuous. On the other hand, a problem whose variables are assumed to be discrete is known as a discrete optimization problem. A problem with mixed variables, that is, both continuous and discrete, is known as a mixed-integer programming problem. In this thesis, we focus on nonlinear global optimization problems with continuous variables.

Global optimization problems can be classified according to the properties of the objective function and constraints. A problem that has no constraints is called an unconstrained global optimization problem. A problem with linear (nonlinear) constraints and nonlinear objective function is called a linearly (nonlinearly) constrained global optimization problem. A problem with constraints given as bounded decision variables only is known as a bound constrained global optimization problem. All these problems often contain functions that are characterised by one or more of the following structures:

- The exact gradient of the objective function (and/or constraints, if they exist) is very expensive to calculate.
- The calculation of the objective function (and/or constraints, if they exist) is time consuming and/or very expensive.
- The objective function (and/or constraints, if they exist) are noisy.

Such problems frequently arise in many real-life applications. There is no method that can solve all these different types of problems. Thus, different methods have been developed to tackle different types of problems. The main idea behind the development of these methods is to develop methods that

- work for a wide range of problems,
- find the global optima with an absolute guarantee,
- are easy to implement, and

- use very little computation.

However, very little has been achieved in finding a method that satisfies all the above mentioned qualities. In most cases, optimization methods are chosen to match the structure of the relevant problems at hand. For example, if the objective function and constraints are differentiable then methods that take advantage of the differentiability of the problem are used [25]. However, if the objective function and/or the constraints are nondifferentiable or the derivatives are very expensive to compute, then methods that do not require the calculation of the derivatives are needed.

In this thesis, we consider unconstrained (or bound constrained) global optimization problems in a continuous space. We propose new hybrid methods that combine some new ideas as well as elements of some existing global optimization methods that have been developed to tackle unconstrained global optimization problems. Specifically, we concentrate on a number of global optimization methods that are known as derivative-free (or direct search) global optimization methods, i.e. methods that do not require any properties of the function being optimized.

1.1 Problem formulation

We consider the problem of finding the global minimum of the unconstrained global optimization problem

$$\min_{x \in \Omega} f(x). \tag{1.1}$$

The objective function $f : \Omega \subset \mathbb{R}^n \rightarrow \mathbb{R}$ is a nonlinear continuous real-valued function. The point x is an n-dimensional vector of continuous variables, that is, $x = (x_1, x_2, \cdots, x_n)^T \in \mathbb{R}^n$. The domain of the search space, Ω, is defined by specifying upper (u^j) and lower (l^j) limits of each jth component of x.

The above optimization problem involves finding the best point x in Ω that gives the least value of the objective function. This point is called a *minimizer* of f over Ω. In dealing with this optimization problem, one has to differentiate between two types of minimizers, *local* and *global* minimizers. A point $x_{loc} \in \Omega$ is called a local minimizer of f over Ω if there exists $\epsilon > 0$ such that $f(x) \geq f(x_{loc}), \forall x \in \Omega$ and $|| \ x - x_{loc} \ || < \epsilon$. On the other hand, a point $x_{opt} \in \Omega$

is said to be a global minimizer of f over Ω if $f_{opt} = f(x_{opt}) \leq f(x), \forall x \in \Omega$. Thus, a global minimizer $x_{opt} \in \Omega$ is the best local minimizer in Ω. Without loss of generality, we consider only minimization problems since maximizing f is similar to minimizing $-f$.

Global optimization methods that solve the optimization problem (1.1) can be classified into two categories: deterministic and stochastic (probabilistic) methods, depending on whether or not they incorporate any stochastic elements. Floudas and Pardalos [35], Horst and Pardalos [50], Horst and Tuy [51], Pardalos and Romeijn [84] and Törn and Zilinskas [102] provide excellent surveys of these methods. Deterministic methods are applicable to optimization problems involving certain mathematical structures. Examples of deterministic methods include branch and bound, Lipschitzian and interval methods [58, 59]. The conditions imposed on deterministic methods mean that they can only be applied on some specific problems. For example, a Lipschitz constant has to be provided for Lipschitzian global optimization methods. On the other hand, stochastic methods do not require such requirements. For example, stochastic methods evaluate the objective function f in a random sample of points from the feasible search region Ω and subsequently manipulate the sample. They attempt to cover the whole search region, so that the global minimum could be identified. They can be applied to ill-structured global optimization problems, for example, problems with "black-box" functions. Examples include, but are not limited to, multi-start methods, level set methods and clustering methods [65, 93, 94, 97, 102]. For stochastic methods, convergence means that the probability for convergence approaches 1 as time goes to infinity, that is, a stochastic method converges to an optimal value with probability 1 as the iterations go to infinity [102]. However, continuing the iteration up to infinity is an impossible task. Thus, the stochastic methods are implemented in finite time, hence compromising the convergence guarantee. Therefore, the finite time implementation of these methods results in heuristic techniques that find near optimal solutions to complex real-life global optimization problems.

1.2 Heuristic techniques

Heuristic techniques [26, 40, 81, 83, 90, 91, 92, 107], also referred to as *meta-heuristics* in the literature, constitute a class of robust computational optimization tools often inspired by the study of natural processes. A common factor in some heuristic techniques is that they combine

some rules and randomness to imitate natural phenomena. They seek good feasible solutions to optimization problems in circumstances where the complexity of the problems or the limited time available do not allow exact solutions to be obtained. Heuristic techniques can be divided into two broad classes: single solution methods and population set-based methods. The methods belonging to single solution methods extensively explore the search region Ω by moving at each step from a current (often random) point to another promising point in its neighbourhood. Often, one of the neighbours becomes the current point, and its neighbours are also explored. Among the single solution methods, simulated annealing [1, 5, 6, 29, 45, 55, 60, 82, 106], tabu search [22, 36, 37, 38, 39, 41, 61], DIRECT method [17, 56] and hit-and-run based methods [3, 110, 111] are well known.

Some single solution methods often accept non-improving points in order to escape being trapped in local minima. For example, in simulated annealing, an annealing process is used to generate new trial points and non-improving trial points are often accepted with some probability. In tabu search, the search is concentrated on more attractive areas which were not earlier explored while setting some points as forbidden. Hit-and-run based methods [110] iteratively generate sequences of points by taking steps of random length in randomly generated directions. The acceptance is often similar to that of simulated annealing, see for example, Hide-and-Seek method [3, 110]. This helps the methods to escape from local minima. The DIRECT method, on the other hand, derives its name from the way it moves towards the global optimum, i.e. 'DIviding RECTangles'. This method is a modification of the Lipschitzian method. It eliminates the need to specify the Lipschitz constant. The method samples points in the domain Ω and uses the information it obtains in deciding which region to search next.

Population set-based methods, by contrast, explicitly work with a population of candidate points (called individuals) randomly sampled all over the search space. The population is stored in a set S. An objective function, called a fitness function, associates each individual with a fitness value (function value) that reflects its quality. Starting with an initial population of individuals, usually generated randomly, these methods try to improve the quality of the individuals by making the population evolve. The evolution is achieved using information exchanges between individuals in order to create new ones or to modify the existing ones. Individuals that exchange information are known as 'parents' and the new individuals created (or modified) are referred to as 'children'. The exchange of information is realised by operators that usually depend on

the considered algorithm. Then selection of individuals (or members) for the next population is done by replacing either all or part (least fittest) of the current population by the children. How the individuals for the next population are selected varies from method to method. The population size can also either be kept constant or changed during the evolution. This process of selection of parents, generation of children and selection of new individuals for the next population is repeated until a certain stopping condition is satisfied. A basic pseudo-code of a population set-based method can be formalized as follows:

Pseudo-code of a population set-based algorithm

Step 1 Generate initial population.

Step 2 Evaluate the fitness of individuals in the population.

Step 3 **while** not stopping criteria **do**

- **Step 3a** Select individuals from the population to be parents
- **Step 3b** Create children
- **Step 3c** Evaluate the fitness of the children
- **Step 3d** Replace all or some individuals in the population by the children

The list of population set-based methods includes, but is not restricted to, genetic algorithms [11, 27, 42, 46, 49, 52, 53, 66, 69, 70, 71, 75, 89, 105], differential evolution [9, 10, 13, 33, 64, 74, 80, 85, 95, 98, 99], controlled random search [4, 7, 18, 44, 73, 86, 87, 88, 101], scatter search [28, 47, 63], electromagnetism-like algorithm [15, 16, 28], ant colony [30, 31, 43, 67] and particle swarm optimization [13, 14, 20, 32, 104]. Although these methods operate globally on a set of points, they differ in the way the populations evolve. For example, in genetic algorithms, the population is evolved using genetic operators such as selection, crossover and mutation. In differential evolution, the population is evolved by using mutation, crossover and acceptance. In controlled random search, the evolution is achieved by replacing the worst point in the set by a new better point. In electromagnetism-like algorithm, the search is directed towards the all-time best point in the set by simulating the electromagnetism theory of physics (Coulomb's Law of electromagnetism) whereas in particle swarm optimization the population evolves by using the concept of bird-flocking and the swarm theory. Scatter search, on the other hand, uses reference sets (from the population set) to combine its points to construct children.

Population set based methods have many advantages compared to the traditional nonlinear programming methods. For example, they

- can be applied to problems that consists of discontinuities, nondifferentiable and nonconvex objective functions,
- do not require the calculation of gradients of the objective function,
- can easily escape from local minima, and
- are easy to implement.

They are, therefore, applicable to a wide range of problems. Consequently, efforts have been made on improving the efficiency and reliability of these methods. In this study, we propose a number of modifications to some of these population set based methods. Most of the modifications are based on hybridization. In particular, we suggest hybrids of de, crs2 and ga with arithmetic crossover, to improve their efficiency and reliability.

1.3 Hybridization

The basic idea behind hybridization is that the combination of principles (elements) from different methods give rise to new methods that display desirable properties of the original methods but not their weaknesses. The aim is to design hybrids that will possess only the strengths of the original methods. There are different ways to hybridize methods [100, 109]. One approach is to run one algorithm until it stops before the next one is started. This is known as sequential hybridization. Another approach is to run the algorithms in parallel in a pre-defined manner or execute the algorithms in a different order per iteration. This is known as parallel hybridization. However, in most cases, hybridization is achieved by combining algorithmic principles of the original methods to end up with a single algorithmic design.

A common way of hybridizing is to combine global techniques of a global method with local techniques of a local method. Local methods use local information around the current point to move in a promising direction. This makes the local methods to be computationally effective. However, they easily get trapped in local minima. On the other hand, global methods explore the whole search region without using any local information. This, in turn, makes them

more reliable in locating the global minimum, that is, they are less likely to get trapped in local minima. Consequently, their computational cost is high. Thus, local methods are good in accuracy and efficiency while they are poor in reliability. By contrast, global methods are poor in accuracy and efficiency while good in reliability. Here, by accuracy we mean how accurate is the final solution obtained by the method in comparison to the exact solution of the problem. Efficiency refers to how the method performs computationally, i.e. in terms of time and the number of function evaluations needed to obtain a solution. By reliability, we mean how successful is the method in finding the global minimum.

It is assumed that the combination of elements from local methods with those from global methods would result in methods that are more efficient, accurate and more reliable in locating the global minimum. In most hybrids, therefore, the strengths of local methods are combined with the strengths of global methods so that the new hybrids possess only the 'merits' from both the local and global methods. Thus, a hybrid method would be more reliable in locating the global minimum while it would also be more accurate and converge faster than a global method. Typical examples include [65], where differential operators are used to generate points for a multi-start global minimization algorithm with dynamic search trajectories [97], and [2], where the search directions for the tabu search are generated using Hooke and Jeeves technique [12]. Other hybrids include [45, 62, 72] for simulated annealing, [24, 52, 79, 109] for genetic algorithms, [28, 103] for continuous scatter search, [23] for continuous tabu search and [33] for differential evolution.

In the next chapters, we concentrate on the work done in the papers **[A]-[E]**. In Chapter 2, paper **[A]**, which describes the first two new de algorithms, is presented. Chapter 3 presents the first four new crs algorithms proposed (paper **[B]**). The third new de algorithm and the fifth new crs algorithm are presented in Chapter 4 (paper **[C]**). The fourth and fifth new de algorithms (paper **[D]**) are presented in Chapter 5. We present paper **[E]**, dealing with genetic algorithms, in Chapter 6. Evaluation of the overall performance of the new algorithms is presented in Chapter 7. Chapter 8 contains the Conclusion.

Chapter 2

Paper [A]: A numerical study of some modified differential evolution algorithms

In **[A]**, we presented two new modified differential evolution (de) algorithms, the differential evolution algorithm with random localization (derl) and the differential evolution algorithm with localization using the best vector (delb). Before introducing the new algorithms, we briefly discuss the features of the original de algorithm. The de algorithm uses a set

$$S = \{x_1, x_2, \cdots, x_N\}$$

of N points, called vectors, initial generated from the search space Ω. The set S evolves per iteration. At each iteration, each vector x_i ($i = 1, 2, \cdots, N$) is targeted to be replaced with a new better vector y_i, called a trial vector. If x_i is targeted for replacement, we call it a target vector. The trial vector y_i is obtained using mutation and crossover.

Mutation is achieved by adding a scaled difference of two vectors to a third vector, called a base vector. The vectors used are randomly selected from the set S and they are mutually different and also different from the target vector x_i. Of the three randomly chosen vectors from S, one is randomly chosen to be the base vector. The new vector $\hat{x}_i$, called the mutant vector, obtained by mutation, is crossed with the target vector x_i to give the trial vector y_i which in turn competes with the target vector. In the original de algorithm, the scaling factor F, used to scale the difference vector in mutation, lie in $[0, 2]$ [99]. There is no optimal value of the scaling factor that has been suggested in the literature. The value of F has been derived empirically and in many cases F varies from 0.4 to 1 [8, 85, 99].

In the crossover phase, a crossover constant $C_R \in [0, 1]$ is used to control the diversity of the trial vector y_i. This allows for the trial vector to have components from the components of the mutated vector $\hat{x}_i$ as well as from the components of the target vector x_i. This parameter is also difficult to choose and is problem dependent. In [99], a higher value, $C_R = 0.9$, is suggested for faster convergence. The effect of C_R was further studied in [8] and it was found that $C_R = 0.5$ is a good choice for many problems in terms of success, and the number of function evaluations needed, in finding the global minimum. Moreover, it was observed that higher values of C_R compromise the success of the algorithm in locating the global minimizer [8].

Mutation and crossover are repeated until all members of S have been targeted. After all N trial vectors have been created, acceptance is applied. In the acceptance phase, the function value of the trial vector y_i is compared with that of the target vector x_i. If $f(y_i) < f(x_i)$ then y_i replaces x_i in S, otherwise S retains x_i. Notice that at each iteration, de attempts to replace all current members of S.

We carried out a numerical study using 50 test problems (see first 50 problems from appendix E) to find 'suitable' values of F and C_R. The de algorithm was run a hundred times on each problem. A run was terminated when either the function values of all vectors in S were identical to an accuracy of four decimal places, i.e

$$|f_h - f_l| \leq \epsilon = 10^{-4}, \tag{2.1}$$

where f_h is the worst function value and f_l is the best function value in S, or the maximum number of iterations ($T = 10000$) was reached. We compare the results using success rate, sr, and average number of function evaluations, fe, of those runs for which the global minima were obtained. The success rate of an algorithm on a problem is the number of successful runs out of a hundred runs. A success was counted when the following condition was met

$$f_l - f_{opt} \leq 0.01, \tag{2.2}$$

where f_{opt} is the known global minimum of the problem and f_l is the best function value obtained when the algorithm terminates. We first study the effect of F by fixing C_R. Table 2.1 shows a summary of results obtained using $F = 0.3, 0.5, 0.75$ and $C_R = 0.5$. The results are only for those problems for which the de algorithm was successful in finding the global minimum. The results from Table 2.1 show that $F = 0.5$ is a good choice both in terms of fe and

sr. For $F = 0.3$, we have less fe but sr is also less. For $F = 0.75$, we have worse fe and lesser sr than those obtained for $F = 0.5$. This, therefore, shows that a value of F less than 0.5 is not very effective in terms of success while a value of F greater than 0.5 produces high fe.

Tab. 2.1: Comparison of total fe and sr using de with $C_R = 0.5$

	F=0.3	F=0.5	F=0.75
fe	1895112	2131089	2200727
sr	3970	4298	4210

We also studied the effect of C_R by fixing F and varying C_R. Table 2.2 shows the results obtained using $C_R = 0.3, 0.5, 0.75$ and $F = 0.5$. From this table, it is clear that $C_R = 0.5$ is a good choice in terms of fe and sr. It shows that increasing C_R reduces fe dramatically. However, this has a very negative effect in sr. On the other hand, for $C_R = 0.3$ and $F = 0.5$, sr is good while fe is poor. In both cases (Table 2.1 and 2.2), we kept the population size and the stopping condition constant. Based on the empirical values of C_R and F obtained by the numerical studies in this thesis and in [8, 99], the values $C_R = 0.5$ and $F = 0.5$ were used for all results obtained using the de algorithm.

Tab. 2.2: Comparison of total fe and sr using de with $F = 0.5$

	$C_R = 0.3$	$C_R = 0.5$	$C_R = 0.75$
fe	2310821	2131089	1904954
sr	4306	4298	4002

We now discuss the new algorithms. The first new algorithm, derl, is based on the randomization of the scaling factor F and the use of the best of the three random vectors used in the mutation phase of de as the base vector. In particular, we randomize F, i.e. $F_i \in [-1, -0.4] \cup [0.4, 1]$ for each target vector x_i and use tournament selection in deciding the base vector. Tournament selection is a process of randomly choosing two or more vectors and obtaining the best within the randomly chosen vectors. The main idea behind this modification is that fixing F throughout a run of the de algorithm has an effect of restricting the exploration. Thus, randomizing F improves exploration. The idea behind the tournament selection of the

base vector is that when the vectors in S cluster around a minimizer, this will have a local effect and consequently improves the convergence rate. Notice that when the vectors in S are scattered around the search region, the tournament selection of the base vector still maintains the exploratory feature of the de algorithm. Thus, a combination of a variable F_i and a tournament selected base vector gives a random localization feature that gradually transforms itself from an exploration (when the vectors in S are scattered) to an intensification feature for rapid convergence (when the vectors in S are clustered around the global minimizer). The crossover constant C_R was set to 0.5.

In the second new version, delb, we adopted a localization after a trial vector has been computed. In the original de algorithm, after a trial vector y_i ($i = 1, 2, \cdots, N$) has been calculated, it competes with the corresponding target vector x_i ($i = 1, 2, \cdots, N$), that is, if $f(y_i) < f(x_i)$ then y_i replaces x_i in S. Otherwise, x_i is retained in S. In delb, we devised a mechanism that explores the regions around the current best vector, x_b, in S and a successful trial vector y_i before acceptance. In particular, each time a trial vector y_i is obtained and it is better than the target vector x_i but worse than the current best vector x_b, two new vectors, r_i and c_i, are obtained, with some probability, using

$$r_i = x_b - (y_i - x_b), \tag{2.3}$$

and

$$c_i = x_b + 0.5(y_i - x_b). \tag{2.4}$$

The vector r_i is a reflection of the trial vector y_i through the current best vector x_b, and the vector c_i is a vector in the middle of y_i and x_b. The best of the vectors y_i and r_i or c_i then competes with the target vector x_i. This has an effect of more exploration in the initial stages of the search and a more exploitation towards the final stages of the search. Here, c_i is calculated if r_i is either worse than y_i or not within the search region.

Equations (2.3) and (2.4) are derived from the following general rules:

$$r_i^j = \alpha^j y_i^j + (1 - \alpha^j) x_b^j \tag{2.5}$$

and

$$c_i^j = \alpha^j x_b^j + (1 - \alpha^j) y_i^j, \, j = 1, 2, \cdots, n, \tag{2.6}$$

where α^j are uniform random numbers in $[-1, 1]$ for each j and n is the dimension of the problem. Equation (2.3) is obtained from equation (2.5) by setting $\alpha^j = -1$ and equation (2.4) is obtained from both equations (2.5) and (2.6) by setting $\alpha^j = 0.5, \forall j$. Thus equations (2.3) and (2.4) are special cases of (2.5) and (2.6) respectively. Equations (2.5) and (2.6) are similar to the arithmetic crossover operator used in real coded genetic algorithms [70] to generate children.

The performance of the methods were evaluated using the first 50 test problems from appendix E. Before deciding which pair of equations, (2.3) and (2.4) or (2.5) and (2.6), to use in delb, we first carried out a numerical study using a small sample of problems from the test problem set. We refer to the delb algorithm using (2.5) and (2.6) as delb1. We used 18 problems, $1, 2, \cdots, 10, 13, 18, 19, 26, 33, 37, 48, 50$, selected according to dimensionality, to evaluate the performance of delb and delb1. Table 2.3 shows a summary of the total results obtained, where we have also included the average cpu times (cpu). The algorithms were run a hundred times on each problem. We realised from the results that the two methods gave a much similar success rate, sr. However, delb was much better in terms of fe and cpu needed to solve the problems. Thus, delb converges faster than delb1 whilst it is still as effective as delb1 in terms of locating the global minimum. Therefore, equations (2.3) and (2.4) were used in delb. Notice that, like in derl, we used a variable F_i in delb along with $C_R = 0.5$.

Tab. 2.3: Comparison of delb and delb1 using total results

Algorithm	fe	sr	cpu
delb	224354	1798	5.44
delb1	281472	1796	8.75

The new algorithms were then compared with the original de algorithm. The algorithms were compared using fe, sr and cpu. We evaluated the efficiency of the algorithms using fe and cpu while the reliability of the algorithms was based on sr. The problems were solved a hundred times with each algorithm and a success was counted as before. The averages were also taken over all successful runs. All the algorithms had positive success rate for 46 out of 50 problems. In all cases where the algorithms failed, they converged to a local minimum.

Included Article

Paper [A]

A Numerical Study of some Modified Differential Evolution Algorithms

P. Kaelo and M.M. Ali

European Journal of Operational Research 169(3), (2006) 1176-1184.

DOI: 10.1016/j.ejor.2004.08.047

Chapter 3

Paper [B]: Some variants of the controlled random search algorithm for global Optimization

In **[B]**, we introduced four new controlled random search (crs) algorithms. All these new algorithms are based on the crs2 algorithm. Like de, crs algorithms also use a population set S of points, initially generated randomly from Ω. In the original crs1 algorithm, the set S is contracted by replacing the current worst point x_h in S with a better point at every iteration. Thus, unlike in de, at each iteration only one trial point is created. The trial point y which competes with the current worst point is found by forming a simplex, using $n+1$ distinct points from S and reflecting one of the points through the centroid of the remaining n points of the simplex. If the trial point obtained fails to replace the current worst point in S, another trial point is obtained using another set of $n+1$ random points from S. Thus, a trial point that is worse than the current worst point in S is discarded. In the crs2 algorithm, the current best point x_b, with best function value in S, is always used to form the simplex in creating the trial point. We referred to this trial point generation scheme of crs2 as P_{gs}. Our numerical study with crs2 suggested that it generates many unsuccessful trial points. Consequently, the crs2 algorithm has a low convergence rate.

In our first modification of the crs2 algorithm, we introduced a local technique, which we called local mutation, after every unsuccessful trial point was obtained. The local technique

makes the crs2 algorithm conducive to convergence in that it creates more successful trial points. This has an effect of reducing the number of function evaluations since more successful points are produced. We denoted this local technique by P_{lm}. The local technique P_{lm} finds a new trial point $\tilde{y}$ by reflecting the unsuccessful trial point y through the current best point x_b in S. In particular, we obtain $\tilde{y}$ by coordinate-wise reflection of y through x_b as follows:

$$\tilde{y}^i = (1 + \omega_i)x_b^i - \omega_i y^i, \tag{3.1}$$

where $\tilde{y}^i$, x_b^i and y^i are the i-th coordinates of $\tilde{y}^i$, x_b^i and y^i respectively and ω_i is a random value in $[0, 1]$ for each i. This technique tends to explore the region around the best point. At the earlier stages, when the points in S are scattered, the local technique has an exploratory feature and it improves convergence when the points are clustered around the global minimizer. We referred to this crs2 algorithm using local technique P_{lm} as a crs algorithm with local mutation or crs(P_{gs}, P_{lm}).

In the second new crs algorithm, we suggested another trial point generation scheme, which we called a global point generation by linear interpolation scheme and denoted it by P_{gl}, to generate trial points together with the simplex scheme P_{gs}. P_{gl} uses the concept of model-based trust region methods [68, 78] to generate trial points. Here, a set Z of $n + 1$ distinct points from S is used to find a model that approximates the objective function around the best point in Z. This model is then minimized with respect to a given radius to give a step s_c which is then added to the best point in Z to give a new trial point. This trial point then competes with the current worst point in S. The scheme P_{gl} is invoked with some probability, per iteration. We introduced a scheme that probabilistically combines the point generation schemes P_{gs} and P_{gl} in this new algorithm. This probabilistic scheme rewards a point generation scheme for producing a successful trial point and penalizes it otherwise. In particular, initial probabilities are assigned to the schemes at the beginning of the algorithm and the probability of a scheme in use is either increased or decreased depending on whether it produced a successful trial point or not. The probability of the other scheme is also adjusted accordingly. This probabilistic scheme is known as a reinforcement-learning scheme [19]. A number of reinforcement-learning schemes have been proposed, both linear and nonlinear (see [19, 76] and references therein). In our experiments, we used the nonlinear reinforcement-learning scheme that was used in [76]. Thus,

a point generation scheme was rewarded using

$$\alpha_k = \alpha_{k-1} + \beta_0 \alpha_{k-1}(1 - \alpha_{k-1}) \tag{3.2}$$

and penalized using

$$\alpha_k = \alpha_{k-1} - \beta_1 \alpha_{k-1}(1 - \alpha_{k-1}), \tag{3.3}$$

where α_k is the probability assigned to a scheme, say P_{gs}, at iteration k. The parameter β_0 ($0 \leq \beta_0 \leq 1$) controls the increment of α_k and the parameter β_1 ($0 \leq \beta_1 \leq 1$) controls the reduction of α_k. We referred to this algorithm as a probabilistic crs using simplex and linear interpolation or crs($P_{gs+gl}, \cdot$). P_{gs+gl} means that the global trial points are generated either by P_{gs} or P_{gl} per iteration. We denote crs2 by crs($P_{gs}, \cdot$) since it uses P_{gs} only. Similarly, we introduce crs($P_{gl}, \cdot$) if P_{gl} replaces P_{gs} in crs($P_{gs}, \cdot$).

Tab. 3.1: Comparison of crs($P_{gs}, \cdot$) and crs($P_{gl}, \cdot$) using 14 problems

Problem		crs($P_{gs}, \cdot$)			crs($P_{gl}, \cdot$)		
P	n	fe	sr	cpu	fe	sr	cpu
ACK	10	17586	100	0.43	7420	100	1.47
B1	2	819	74	0.01	420	100	0.01
B2	2	825	76	0.01	423	100	0.01
CB3	2	554	97	0.01	291	100	0.01
CM	4	2266	100	0.02	1307	100	0.04
EXP	10	6425	100	0.15	2958	100	0.60
GW	10	10037	100	0.25	4429	100	0.87
LM2	10	8945	100	0.22	21085	100	2.63
MGP	2	623	25	0.01	20000	100	0.29
PRD	2	497	100	0.01	253	100	0.01
PQ	4	2189	100	0.02	1497	100	0.05
RG	10	100000	0	2.17	5524	100	1.13
SAL	10	45060	0	0.95	12187	100	2.21
tr		219893	972	4.81	177794	1300	14.56

We introduced the probabilistic scheme, P_{gs+gl}, after we realised from numerical experiments that P_{gl} was good on certain functions and bad on others. In particular, it proved successful on functions where the simplex scheme P_{gs} had difficulties. On the other hand, it performed poorly in terms of fe on certain functions that P_{gs} had no difficulties. To show the performance of crs($P_{gs}, \cdot$) and crs($P_{gl}, \cdot$), we present some results in Table 3.1 using 13 randomly selected problems from the test set. We used the stopping condition (2.1) or the maximum iterations

$T = 1000n^2$ to terminate a run. A success was also counted as before, i.e. using (2.2). We use fe, sr and cpu (defined as before) to compare the algorithms. We see from this table that for the functions Ackley (ACK), Exponential (EXP) and Griewank (GW), crs($P_{gs}, \cdot$) needed much more fe than crs($P_{gl}, \cdot$) to reach the global minimum. On the other hand, for the 10 dimensional Levy function (LM10) and the 2 dimensional Multi-Gaussian function (MGP), crs($P_{gl}, \cdot$) needed much more fe than crs($P_{gs}, \cdot$) to reach the global minimum. We also have that for the MGP function, crs($P_{gl}, \cdot$) needed maximum iterations to reach the global minimum. crs($P_{gl}, \cdot$) also needed more cpu. However, in terms of sr, crs($P_{gl}, \cdot$) achieved more sr than crs($P_{gs}, \cdot$). We also presented, in Table 3.1, results of problems that were solved by crs($P_{gl}, \cdot$) but not by crs($P_{gs}, \cdot$). In the case of the Rastrigin (RG) function, crs($P_{gs}, \cdot$) reached the maximum iterations before locating the global minimum and for the Salomon (SAL) function, it converged to a local minimum. crs($P_{gl}, \cdot$) was able to find the global minimum 100 times for these problems. The combined scheme, P_{gs+gl}, solved all these problems very efficiently. Thus, the scheme P_{gs+gl} proved its usefulness.

In the third modification, we introduced the local mutation P_{lm} in crs($P_{gs+gl}, \cdot$), motivated by the results of crs(P_{gs}, P_{lm}). We referred to this algorithm as a probabilistic crs using simplex and linear interpolation with local mutation or crs(P_{gs+gl}, P_{lm}). The local mutation was introduced so as to help the simplex scheme produce successful trial points as in crs(P_{gs}, P_{lm}). Thus, the local mutation is only used alongside the simplex scheme P_{gs}, that is, whenever P_{gs} produces an unsuccessful trial point, local mutation is done. If either P_{gs} or local mutation produces a successful trial point, then the simplex scheme is rewarded. Otherwise it is penalized. We also rewarded P_{gl} for producing a successful trial point and penalized it otherwise.

In the fourth modification, we proposed a probabilistic adaptation of quadratic and linear interpolation. The quadratic interpolation scheme P_{gq} was proposed in the literature [7, 73] as a substitution of the simplex scheme P_{gs} in generating trial points in crs2. In this scheme, two random points from the set S together with the best point x_b in S are used to find a trial point by coordinate-wise quadratic interpolation. This scheme is very efficient in terms of the number of fe and cpu but is inferior in terms of sr [8]. We thus proposed a probabilistic adaptation of P_{gq} and P_{gl} so as to improve its reliability. We referred to this algorithm as a probabilistic crs using quadratic and linear interpolation or crs($P_{gq+gl}, \cdot$).

The performance of the new methods was judged using a set of 50 test problems (the same

problems used in **[A]**). The new methods were compared with crs($P_{gs}, \cdot$), crs($P_{gs}, P_{l\beta}$) and crs($P_{gq}, \cdot$), where P_{gq} replaces P_{gs} in crs($P_{gs}, \cdot$). crs($P_{gs}, P_{l\beta}$) [4] uses a β-distribution as a local technique in crs($P_{gs}, \cdot$) every time a new best point is obtained by P_{gs}. Of the 50 test problems used, all the algorithms failed to solve four 10 dimensional functions Epistatic Michalewicz (EM), Odd Square (OSP), Schwefel (SWF) and Shekel's Foxholes (SF). Therefore, these functions were excluded from the performance evaluation.

Before the performance evaluation of the algorithms, a numerical study was first carried out to determine the best combination of the parameter values β_0 and β_1 for the probabilistic crs algorithms crs($P_{gs+gl}, \cdot$), crs(P_{gs+gl}, P_{lm}) and crs($P_{gq+gl}, \cdot$). Several combinations of β_0 and β_1 were used to determine suitable values of these parameters. We observed that these values are problem dependent. Our numerical experiments with β_0 and β_1 suggested that values of (β_0, β_1) with $\beta_0 < \beta_1$ produced overall best results. This means we penalized a scheme more for failing while rewarding it less for succeeding (see equations (3.2) and (3.3)). Table 3.2 presents a summary of some results obtained using crs($P_{gs+gl}, \cdot$) for a number of combinations of β_0 and β_1. The results are the totals for all problems, out of 50 problems, that crs($P_{gs+gl}, \cdot$) managed to solve. It is clear from the table that the results obtained for values of (β_0, β_1) with $\beta_0 < \beta_1$ are almost the same. Better fe and sr were produced when $\beta_0 < \beta_1$. On the other hand, the results for values of (β_0, β_1) with $\beta_0 > \beta_1$ are slightly higher in fe and less in sr. Overall best results, on average, were obtained using the combination $\beta_0 = 0.35$ and $\beta_1 = 0.65$. Therefore, results reported in **[B]** were obtained by setting $\beta_0 = 0.35$ and $\beta_1 = 0.65$.

Tab. 3.2: Total results of crs($P_{gs+gl}, \cdot$) using 2 combinations of β_0 and β_1

Algorithm	β_0	β_1	fe	sr	cpu
crs($P_{gs+gl}, \cdot$)	0.15	0.70	358928	4059	43.21
crs($P_{gs+gl}, \cdot$)	0.45	0.65	357041	4006	45.42
crs($P_{gs+gl}, \cdot$)	0.65	0.25	375419	3862	48.13

Included Article

Paper [B]

SOME VARIANTS OF THE CONTROLLED RANDOM SEARCH ALGORITHM FOR GLOBAL OPTIMIZATION

P. KAELO AND M.M. ALI

Submitted to Journal of Optimization Theory and Applications.

Chapter 4

Paper [C]: Probabilistic adaptation of point generation schemes in some global optimization algorithms

In **[B]**, we found out that the point generation scheme P_{gs} of crs2 was not reliable in terms of finding the global minimum while its efficiency in terms of fe and cpu was very good. On the other hand, results obtained in **[A]** indicated that the point generation scheme of de was very robust in terms of locating the global minimum but its weakness was its efficiency in terms of fe and cpu. In **[C]**, we combined the complementary strengths of these two point generation schemes.

We introduced two new hybrid algorithms in **[C]**, one for crs2 and the other for de. In both hybrids, we probabilistically embedded the point generation scheme of one algorithm into the other algorithm. That is, we embedded the crs2 point generation scheme, P_{gs}, in de so that the trial points in de are generated using either the crs2 point generation scheme or its own scheme, denoted by P_{gd}. Likewise, we embedded the de point generation scheme P_{gd} in crs2. We also incorporated local techniques in both the algorithms.

In the new crs algorithm, we proposed the local technique P_{lm} described by (3.1). We referred to the new crs algorithm obtained as a probabilistic crs2 with localizations or crs(P_{gsd}, P_{lm}). Here, P_{gsd} means that trial points are generated probabilistically using either P_{gs} or P_{gd}. The local mutation P_{lm} was applied once a trial point generated failed to replace the current worst

point in S. Notice that unlike in **[B]**, where the local technique P_{lm} was only applied when a trial point generated by P_{gs} failed, here it was applied whenever a trial point generated by either scheme, i.e. P_{gs} or P_{gd}, failed.

The point generation scheme P_{gsd} was also used in the new de algorithm. However, unlike in crs(P_{gsd}, P_{lm}), where the scheme generated one trial point per iteration, in the new de algorithm it was used to generate N trial points per iteration. For localization, we proposed the local technique described by (2.3) - (2.4) and denoted it by P_{lb}. The local technique P_{lb} was invoked in the acceptance phase as was done in delb in **[A]**. We referred to the new de algorithm obtained as a probabilistic de with localizations or de(P_{gsd}, P_{lb}).

The performance of the methods in **[C]** was evaluated using the same 50 test problems that were considered in **[A]** and **[B]**. The algorithms were run a hundred times. We used fe, sr and cpu, defined as before, for the comparison. For the de type algorithm, we used the stopping condition that was used for the de type algorithms in **[A]**, i.e. (2.1) and $T = 10000$. Similarly, for the crs type algorithm, we used the stopping condition that was used for other crs type algorithms in **[B]**, i.e. (2.1) and $T = 1000n^2$. A success was also counted as before.

Like in **[B]**, we first carried out some numerical study using randomly selected problems to determine the values of the parameters used in both the crs(P_{gsd}, P_{lm}) algorithm and the de(P_{gsd}, P_{lb}) algorithm. These parameters are β_0 and β_1, common to both the crs and de type algorithms, and the parameter R_c, used in the de type algorithm only. The parameter R_c is the number of points (vectors) replaced per iteration in the de type algorithm. Another parameter used in de(P_{gsd}, P_{lb}) is the parameter ω which controls the frequency of localizations in the acceptance phase. This parameter was studied in **[A]** and it was found that $\omega = 0.1$ is a good value to use. Therefore, this value was also used in **[C]**.

We used 15 randomly selected problems, $1, 13, 15, 18, 19, 23, 31, 33, 35, 43, 45, 46, 48, 49, 50$, from appendix E in our numerical study to determine the values of the parameters β_0, β_1 and R_c. For the purpose of this numerical study, we re-numbered the above 15 problems as $1, 2, \cdots, 15$ respectively. We used crs($P_{gsd}, \cdot$) to determine the parameter values for the crs type algorithm and de($P_{gsd}, \cdot$) to determine the parameter values for the de type algorithm. The crs($P_{gsd}, \cdot$) and de($P_{gsd}, \cdot$) algorithms are crs(P_{gsd}, P_{lm}) and de(P_{gsd}, P_{lb}), respectively, without local technique. We used crs($P_{gsd}, \cdot$) since assigning a probability to the local technique in crs(P_{gsd}, P_{lm}) would force the algorithm to quickly go to a local minimum. As for the de type algorithm, we did

not use de(P_{gsd}, P_{lb}) because the probabilistic adaptation is carried out independently of the localization. Hence it was sufficient to use de($P_{gsd}, \cdot$). We present results of the crs($P_{gsd}, \cdot$) algorithm obtained using two combinations of (β_0, β_1, C_R) in each of the figures 4.1 and 4.2. In both figures, we only present the fe obtained. In the figures, the horizontal axis represents the functions and the vertical axis represents fe.

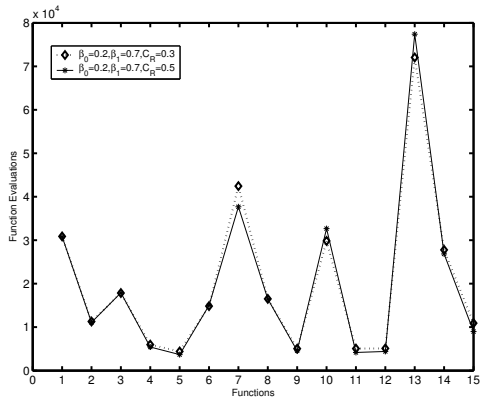

Fig. 4.1: Variations of C_R for $\beta_0 = 0.2$ and $\beta_1 = 0.7$ on crs($P_{gsd}, \cdot$)

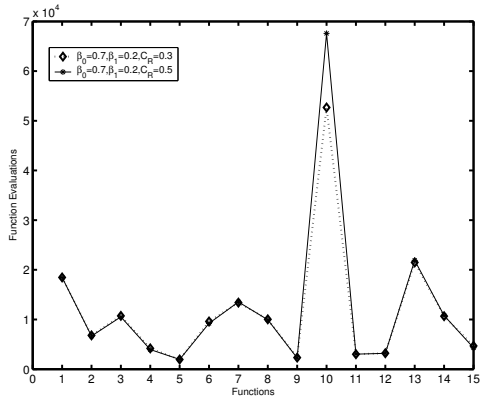

Fig. 4.2: Variations of C_R for $\beta_0 = 0.7$ and $\beta_1 = 0.2$ on crs($P_{gsd}, \cdot$)

Figure 4.1 presents the number of function evaluations for two sets of parameter values, i.e. $(\beta_0, \beta_1, C_R) = (0.2, 0.7, 0.3)$ and $(\beta_0, \beta_1, C_R) = (0.2, 0.7, 0.5)$. Likewise, figure 4.2 presents fe for two sets, $(\beta_0, \beta_1, C_R) = (0.7, 0.2, 0.3)$ and $(\beta_0, \beta_1, C_R) = (0.7, 0.2, 0.5)$. Notice that in figure 4.1, we kept $(\beta_0, \beta_1) = (0.2, 0.7)$ fixed and varied C_R. Similarly, in figure 4.2, we kept $(\beta_0, \beta_1) = (0.7, 0.2)$ fixed and varied C_R. Figure 4.1 shows that except for two functions, functions 7 and 13, the values $C_R = 0.3$ and $C_R = 0.5$ gave about the same fe. Likewise, figure 4.2 shows that apart from one function, function 10, the fe obtained for each function

is also about the same. A comparison of figures 4.1 and 4.2 shows that $\beta_0 = 0.7$ and $\beta_1 = 0.2$ produce a much lower fe than that produced by $\beta_0 = 0.2$ and $\beta_1 = 0.7$. However, the combination $\beta_0 = 0.2$ and $\beta_1 = 0.7$ achieved about 94% success rate compared to about 85% success rate obtained by $\beta_0 = 0.7$ and $\beta_1 = 0.2$ on the same problems. Notice that the successes are not shown in the figures. We also observed from numerical experiments that values of (β_0, β_1) with $\beta_0 < \beta_1$ gave, on average, better results. Thus, in our experiments, we used combinations that had β_0 less or equal to β_1. Results of crs(P_{gsd}, P_{lm}) presented in **[C]** are for the combination $\beta_0 = \beta_1 = 0.15$ and $C_R = 0.3$. This combination was empirically found to be superior to the other combinations considered.

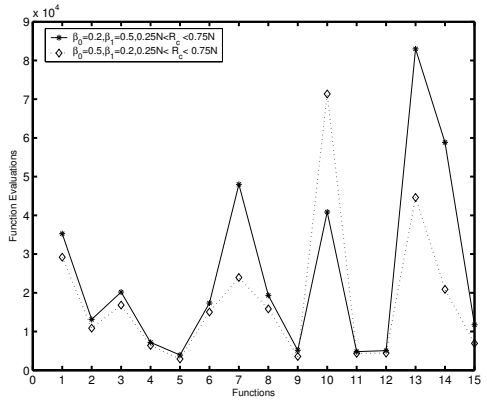

Fig. 4.3: Varying β_0 and β_1 in de($P_{gsd}, \cdot$); R_c constant

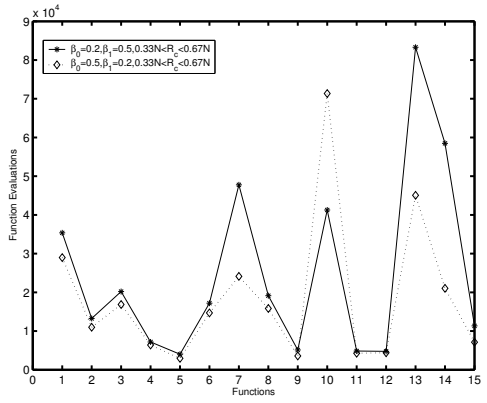

Fig. 4.4: Varying β_0 and β_1 in de($P_{gsd}, \cdot$); R_c constant

We also present results of the de($P_{gsd}, \cdot$) algorithm obtained using two combinations of (β_0, β_1, R_c). These results are presented in figures 4.3-4.4. In both figures, we only present fe obtained by different combinations for these parameter values. The horizontal axis in the

figures represents the functions and the vertical axis represents fe. The results were obtained by varying β_0 and β_1. These parameter values, (β_0, β_1), were set to $(0.2, 0.5)$ and $(0.5, 0.2)$ in both figures. Results in figure 4.3 were obtained by penalizing the scheme, i.e. P_{gs} or P_{gd}, used if $R_c \leq 0.25N$ (nearest integer) and rewarding it if $R_c \geq 0.75N$ (nearest integer), where N is the population size of set S. Neither scheme was rewarded nor was penalized if $0.25N < R_c < 0.75N$. In figure 4.4, a scheme was penalized if $R_c \leq 0.33N$ (nearest integer) and rewarded if $R_c \geq 0.67N$ (nearest integer). We also did not penalize nor reward any scheme if $0.33N < R_c < 0.67N$. From figures 4.3 and 4.4, we see that apart from one function, function 10, the de($P_{gsd}, \cdot$) algorithm with a combination of $\beta_0 = 0.2$ and $\beta_1 = 0.5$ has much higher fe. However, the combination of $\beta_0 = 0.5$ and $\beta_1 = 0.2$ had a lesser success rate than the combination of $\beta_0 = 0.2$ and $\beta_1 = 0.5$. As in crs($P_{gsd}, \cdot$), we observed that combinations of β_0 and β_1 with $\beta_0 < \beta_1$ produced, on average, better results with respect to sr, fe and cpu. In **[C]**, we reported results of de(P_{gsd}, P_{lb}) obtained using $\beta_0 = 0.30$ and $\beta_1 = 0.45$. These values were found to produce better results than the other combinations considered. We penalized a scheme if $R_c \leq 0.33N$ and rewarded it if $R_c \geq 0.67N$.

Included Article

Paper [C]

PROBABILISTIC ADAPTATION OF POINT GENERATION SCHEMES IN SOME GLOBAL OPTIMIZATION ALGORITHMS

P. KAELO AND M.M. ALI

Optimization Methods and Software, in press.

DOI: 10.1080/10556780500094671

Chapter 5

Paper [D]: Differential evolution algorithms using hybrid mutation

In **[D]**, two other hybrids of de were introduced by embedding a point generation scheme that uses an attraction-repulsion concept of electromagnetism-like algorithm [15, 16]. These versions of de only modify the mutation rule of de. The new point generation scheme, denoted by M_e, incorporated in de is a variant of electromagnetism-like (em) scheme. M_e is a variant of electromagnetism-like scheme because it differs slightly from the scheme used in [15, 16]. In these new de algorithms, mutant vectors are generated using either the scheme M_e or the original de mutation scheme, which we denoted by M_d. Unlike in **[C]**, these versions of de do not use any probabilistic combination of these trial point generation schemes nor do they implement any local technique. The criterion used to determine whether to use M_d or M_e is based on component-wise standard deviation of vectors in the population set S. That is, the standard deviation σ_k^j of the jth ($j = 1, 2, \cdots, n$) components of vectors in S at the beginning of each kth iteration is calculated. The kth standard deviation is then compared with the initial standard deviation.

In the first hybrid, for each target vector x_i, $i = 1, 2, \cdots, N$, if a randomly chosen σ_k^j is less than a certain percentage of the initial corresponding standard deviation σ_0^j, i.e. $\sigma_k^j \leq \epsilon_1 \sigma_0^j$, then the scheme M_e is invoked in creating the corresponding mutant vector $\hat{x}_i$. Otherwise the scheme M_d is used in creating $\hat{x}_i$. The parameter $\epsilon_1 \in (0, 1)$ controls the frequency of invoking M_e. Notice that both M_e and M_d may be invoked per iteration. The creation of the

corresponding trial vector y_i remains the same as done in the original de algorithm. We referred to this algorithm as the de algorithm with integrated mutation per iteration or deim.

In the second hybrid, only one mutation scheme is used per iteration. The standard deviation is also used to switch between M_e and M_d. However, unlike in deim, we compared $||\sigma_k||$ to $||\sigma_0||$ in determining which scheme to use. Thus, at each iteration k, if $||\sigma_k|| \leq \epsilon_2||\sigma_0||$, then M_e is used in creating N mutant vectors. Otherwise M_d is used in creating the N mutant vectors for the kth iteration. Crossover is also done in the same way as done in the original de algorithm. We referred to this de algorithm as the de algorithm with adapted mutation per iteration or deam.

In **[A]**, it was shown that $C_R = 0.5$ is a good choice when comparing de algorithms on a large set of problems. However, for faster convergence, a higher value of C_R is recommended whilst for a better success a lower value of C_R is recommended [99]. Also, in [8], an intensive numerical study of C_R was done. The study showed that suitable values of C_R lie in $[0.5, 0.75]$. Therefore, in deim and deam, unlike in the previous de algorithms where C_R was kept unchanged throughout the iterations, we varied it per iteration. In particular, we randomly chose C_R in $[0.5, 0.7]$ at every iteration. This has an effect of improving the success rate as well as the convergence rate of the algorithms. We also used a random scaling factor F_i in M_d, see paper **[C]**. Other parameters introduced in the new de algorithms are ϵ_1 and ϵ_2, which control the invoking of M_e in deim and deam respectively. Our numerical study showed that values of $\epsilon_1 \in (0, 0.4]$ had a positive effect of reducing fe without (negatively) affecting the robustness of the algorithms. Similarly, we observed that values of $\epsilon_2 \in (0, 0.4]$ produced better results in terms of fe and cpu without compromising sr of de. For values of both ϵ_1 and ϵ_2 higher than 0.4, fe reduced greatly as did the success rate of the algorithms. Overall best values were empirically found to be $\epsilon_1 = 0.35$ and $\epsilon_2 = 0.4$ for deim and deam, respectively. Therefore, for our numerical experiments, these values were used.

The performance of the algorithms was evaluated using 50 test problems. However, unlike in the previous papers, only 46 problems from appendix E were used. Other four problems were obtained by variations of some problem dimensions. A comparison was made between the de algorithm and the new de algorithms, deim and deam. We compared the algorithms using sr, fe and cpu. As in the previous de type algorithms, these algorithms were run a hundred times and the same stopping conditions applied, i.e. (2.1) and $T = 10000$. A success was also counted as before, i.e. using (2.2).

Included Article

Paper [D]

DIFFERENTIAL EVOLUTION ALGORITHMS USING HYBRID MUTATION

P. KAELO AND M.M. ALI

Submitted to Computational Optimization and Applications.

Chapter 6

Paper [E]: Integrated crossover rules in real coded genetic algorithms

In **[E]**, we suggested five versions of the real coded genetic algorithm with arithmetic crossover. Real coded genetic algorithms have been used in recent years to solve continuous optimization problems [21, 46, 48, 69, 70]. Traditionally, points (chromosomes) in genetic algorithms are represented using binary coded strings. In real coded genetic algorithms, points are represented as real valued numbers. As in binary coded genetic algorithms, real coded genetic algorithms also use crossover operators to cross-bred points (parents) and mutation operators to mutate the elements of the new points (children) created by crossover operators. The introduction of real valued numbers in genetic algorithms has a number of advantages over binary coding. Real coded genetic algorithms are easier to hybridize with other optimization methods and are faster than the traditional binary coded genetic algorithms in terms of convergence [69, 70, 109]. They are also applicable to a wide range of problems with high dimensions. Binary coded genetic algorithms have a problem of precision when applied to problems with very high dimensions and/or with very big search spaces [70].

In **[E]**, we study the real coded genetic algorithm suggested in Michalewicz [70] and also studied in Hu et. al. [54]. This real coded genetic algorithm uses an arithmetic crossover rule for its cross breeding. In **[E]**, we referred to this crossover rule as c_{r1} and the corresponding genetic algorithm as ga$(c_{r1}, \cdot)$. We suggested some modifications to this genetic algorithm to improve its efficiency in terms of the number of function evaluations and reliability in terms of

locating the global minimum.

In the first modification, we suggested a local search technique to the genetic algorithm to improve its exploration at earlier stages of the search and to expedite convergence at later stages. The local technique is invoked whenever the genetic algorithm gives a new best point that is better than the current best in the population set S. We referred to the new real coded genetic algorithm obtained as a genetic algorithm with local technique or ga(c_{r1}, ℓ).

In the second version, we suggested a modification to the crossover rule c_{r1} of the genetic algorithm. We introduced an integrated crossover rule that generates two children at a time using three points (parents). Each parent is selected from the set S by tournament selection of two. In this integrated crossover rule, one child is created using a Nelder-Mead [77] 'simplex-like' crossover rule suggested in Kalganova and Strechen [57]. The same parents are then mated to create another (second) child using the mutation rule of de. We referred to this integrated crossover rule as c_{r2}. The new algorithm is referred to as a genetic algorithm with integrated crossover rule or ga($c_{r2}, \cdot$). We also incorporated the local technique in this algorithm to have the third algorithm. We referred to this algorithm as a genetic algorithm with integrated crossover rule and local technique or ga(c_{r2}, ℓ).

We then probabilistically combined (as done in **[B]** & **[C]**) the crossover rules c_{r1} and c_{r2} to obtain a new probabilistically adapted crossover rule that uses the two crossover rules to generate children. We referred to this new probabilistically adapted crossover rule as c_{r12} and the resulting fourth version of the genetic algorithm as the genetic algorithm based on probabilistic adaptation of crossover rules or ga($c_{r12}, \cdot$). Motivated by the results obtained by the previous two genetic algorithms, ga(c_{r1}, ℓ) and ga(c_{r2}, ℓ), that incorporated the local technique, we incorporated the local technique in ga($c_{r12}, \cdot$) to obtain the fifth algorithm. We referred to this algorithm as a ga($c_{r12}, \cdot$) algorithm with local technique or ga(c_{r12}, ℓ).

The rewarding and penalizing of the crossover rules in the probabilistic genetic algorithms were done as before, using the schemes (3.2) and (3.3). However, to determine whether to penalize c_{r1} (reward c_{r2}) or reward c_{r1} (penalize c_{r2}), we used a scheme that assigns a weight to each child generated depending on its fitness. We used a scheme similar to the one suggested in [108], which was used for parent selection for the mating pool (see **[E]**), to assign weights to the children.

The performance of the new genetic algorithms described above was evaluated using a set

of 50 test problems. The reliability and efficiency of these genetic algorithms were judged based on sr, and fe and cpu, respectively. The sr, fe and cpu were calculated as before and the maximum number of iterations T was set to 10000. Like in other probabilistic algorithms suggested in the previous papers, the probabilistic genetic algorithms, ga($c_{r12}, \cdot$) and ga(c_{r12}, ℓ), have parameters β_0 and β_1, provided by the user, that control the increment and reduction of the probability α_k (3.2-3.3), respectively. We used the ga($c_{r12}, \cdot$) algorithm to determine the best combinations of β_0 and β_1. We observed from numerical experiments that these parameters are problem dependent. However, for most problems in the test suite, we observed that combinations of smaller values of both β_0 and β_1 gave favourable results. Therefore, in our numerical experiments, we used smaller values for both parameters.

Tab. 6.1: Total results of GA($c_{r12}, \cdot$) for different combinations of β_0 and β_1

GA($c_{r12}, \cdot$)			
fe	sr	β_0	β_1
132173	3572	0.2	0.3
146464	3575	0.3	0.45

In Table 6.1, we present a summary of results obtained using some combinations of β_0 and β_1. We present results obtained using higher values of β_0 and β_1 than those used in **[E]** to show that higher parameter values give worse results. We only present the total fe and sr (total averages) for $(\beta_0, \beta_1) = (0.2, 0.3)$ and $(\beta_0, \beta_1) = (0.3, 0.45)$. These results presented here are for the problems shown in Table 1 in **[E]**, excluding the Epistatic Michalewicz (EM), Meyer (MR), Extended Rosenbrock (RB) and Schwefel (SWF) problems. These problems were not used for the combination of parameter values given above. We see from Table 6.1 that results obtained using $\beta_0 = 0.2$ and $\beta_1 = 0.3$ are better than the results obtained using the other combination in terms of fe. However, in terms of sr, both combinations gave about the same sr. Results reported in **[E]** are for the combination $\beta_0 = 0.05$ and $\beta_1 = 0.065$.

Included Article

Paper [E]

INTEGRATED CROSSOVER RULES IN REAL CODED GENETIC ALGORITHMS

P. KAELO AND M.M. ALI

European Journal of Operational Research, in press.

DOI: 10.1016/j.ejor.2005.07.025

Chapter 7

Overall performance evaluation of the new algorithms

This chapter further evaluates the work done in the papers **[A]-[E]**. We start with the evaluation of the new de algorithms proposed in **[A]**.

An analysis of the results, presented in Table 1, in **[A]** shows that in terms of efficiency, using total results, derl is the best performer. In terms of reliability, delb is the best performer, with much higher sr than de and derl. Both the algorithms (derl and delb), however, produced much superior results than those produced by the original de algorithm. Moreover, had we used accuracy as the criterion of comparison, derl and delb would have been much superior to de. For example, for the 4 dimensional Neumaier 2 (NF2), the 10 dimensional Extended Rosenbrock (RB) and the 9 dimensional Storn's Tchebychev (ST) problems, whose global minimum values are zeros, the accuracy of the final solutions found using de were worse than 0.0001, in all cases, whereas the minima obtained by the new algorithms were much accurate. Similar trends were noticed for other problems. The new algorithms obtained minima that were very close to the actual global minima for all the problems that they solved. This shows that the random localization feature, introduced in derl, has a great effect in improving the convergence rate of de. On the other hand, the localization in the acceptance phase, introduced in delb, shows a great effect in improving the reliability of the original de algorithm.

An analysis of the results, presented in Table 1, in **[B]** shows that the probabilistic crs algorithms using simplex and linear interpolation are the best in terms of sr and fe. For example,

in terms of sr, crs(P_{gs+gl}, P_{lm}) is the best with a success rate of 85% (4250 successes out of 5000 runs) and crs($P_{gs+gl}, \cdot$) is the runner-up with 81%. In terms of fe, crs(P_{gs+gl}, P_{lm}) is still the best, with about 59.7% lesser fe than crs($P_{gs}, \cdot$) and about 4.8% better than the runner-up, crs(P_{gs}, P_{lm}). They also solved more problems than crs($P_{gs}, \cdot$), crs(P_{gs}, P_{lm}), crs($P_{gs}, P_{l\beta}$), crs($P_{gq}, \cdot$) and crs($P_{gq+gl}, \cdot$). However, in terms of cpu, the probabilistic crs algorithms faired badly. The same trend is true for the probabilistic crs using quadratic and linear interpolations, crs($P_{gq+gl}, \cdot$). The crs($P_{gs+gl}, \cdot$), crs(P_{gs+gl}, P_{lm}) and crs($P_{gq+gl}, \cdot$) algorithms are worse performers in terms of cpu. The local mutation also showed great improvements in terms of fe, sr and cpu in all algorithms in which it was incorporated. The overall best crs algorithm, therefore, was found to be crs(P_{gs+gl}, P_{lm}). We also tried a probabilistic crs using simplex and quadratic interpolation, i.e. crs($P_{gs+gq}, \cdot$). However, the results for this algorithm are much inferior to the other algorithms that use quadratic interpolation, i.e. crs($P_{gq}, \cdot$) and crs($P_{gq+gl}, \cdot$).

A comparison of the best crs algorithm proposed, crs(P_{gs+gl}, P_{lm}), was made with the DIRECT algorithm developed by Jones et. al. [56]. The DIRECT algorithm is a heuristic search method. It was shown to be very competitive with other existing search methods. We used 30 problems from appendix E for this comparison. For a fair comparison of the algorithms, we used the same stopping condition in crs(P_{gs+gl}, P_{lm}) that was used in the DIRECT algorithm. The analysis of the results showed that the algorithms are very competitive both in fe and sr for low dimensional problems. However, for high dimensional problems, the crs(P_{gs+gl}, P_{lm}) algorithm is superior both in terms of fe and sr.

Overall analysis of the results in **[C]** shows that the new methods proposed, i.e. crs(P_{gsd}, P_{lm}) and de(P_{gsd}, P_{lb}), are much superior to their original counterparts, i.e. crs2 and de, respectively. The de(P_{gsd}, P_{lb}) algorithm achieved about 38% lesser total fe than the de algorithm for about the same sr. On the other hand, crs(P_{gsd}, P_{lm}) achieved about 13% more success in locating the global minimum than the crs2 algorithm.

The results obtained in **[D]** show that the new algorithms suggested are much superior to de both in terms of total fe and cpu. The deim algorithm is better than de by about 44% and 30%, respectively, in terms of total fe and cpu. On the other hand, deam is better than de by about 45% and 40% in terms of total fe and cpu, respectively. In terms of sr, all the algorithms compete very well with each other. However, the overall best performer in all respects is the deam algorithm.

A comparison of the results, presented in Tables 1 and 2, in **[E]** shows that, in terms of sr, the new algorithms are more reliable than the original genetic algorithm which uses the arithmetic crossover rule to generate children. The new genetic algorithms are also slightly better than their original counterpart in terms of efficiency. A comparison of all the new algorithms in **[E]** shows that ga(c_{r2}, ℓ) and ga(c_{r12}, ℓ) are the most reliable, in terms of success rate.

7.1 Comparison of the new algorithms

A number of modifications to de, crs2 and ga were discussed. In this section, we compare all these new algorithms to see which is the best. We compare the algorithms using the results of those functions, from the main tables in respective articles, for which all the methods succeeded in finding their global minimum. That is, we exclude the results of all algorithms for a problem if at least one of the algorithms has sr $= 0$. We start with the comparison of all the new de algorithms.

Tab. 7.1: Comparison of the new de algorithms using total results

Algorithm	fe	sr	cpu
derl	1306848	4235	36.42
delb	1550571	4273	42.38
de(P_{gsd}, P_{lb})	784554	4222	30.25
deim	1056240	4210	38.48
deam	1039312	4250	33.67

Tab. 7.2: Average performance of the new de algorithms based on individual problems

	derl	delb	de(P_{gsd}, P_{lb})	deim	deam
mfe_{rel}	23	6.5	10	25	25
σ_{rel}	10	13	43	24	24
sr	94.2	96.6	94.4	96.2	97

In papers **[A]**, **[C]** and **[D]** we developed different versions of the de algorithm. We now compare all these new versions suggested. We compare the algorithms based on the common

test problems used in the numerical experiments. We present the summarized results in Table 7.1. A comparison of the algorithms using the results from Table 7.1 shows that de(P_{gsd}, P_{lb}) is the most efficient algorithm in terms of fe and cpu. The deam algorithm is the runner-up. In terms of the ability of the algorithms to locate the global minimum, the delb algorithm is the winner and deam is again the runner-up. However, the difference between the sr of all the algorithms is very minimal. Thus, all the algorithms are very competitive in terms of finding the global minimum.

The algorithms presented in **[A]**, **[C]** and **[D]** have, so far, been compared using the total results. However, the total results are mainly dominated by the results of high dimensional problems. We, therefore, compare the algorithms, again, using the percentage fe improvement, fe_{rel}, by the new de algorithms over the original de algorithm. We calculate fe_{rel} for each of the 46 problems where all the algorithms have a positive success rate. We then calculate the mean, mfe_{rel}, and the standard deviation, σ_{rel}, for the 46 problems. Table 7.2 presents mfe_{rel} together with the corresponding standard deviation, σ_{rel}, for the new de algorithms. We also present the percentage success, sr, for each algorithm in Table 7.2, where sr is obtained by dividing the total sr of an algorithm by the total number of problems (46 problems). In terms of mfe_{rel}, the results show that the average fe improvement due to derl and delb is about 23% and 6.5%, respectively. The low standard deviations of about 10% and 13%, respectively, for derl and delb, show that the performance improvement for all the problems is very much the same and not very much problem dependent. On the other hand, mfe_{rel} for de(P_{gsd}, P_{lb}) shows that the average performance improvement is about 10%, with a standard deviation of about 43%. This shows that the performance improvement is problem dependent. The other two algorithms, deim and deam, achieved the same performance improvement, about 25%, with a standard deviation of about 24%. In terms of sr, the new algorithms are very much competitive with the original de algorithm (de has 93% sr).

We do similar comparisons of the new crs algorithms developed. The first comparison is made, in Table 7.3, using the total results. From Table 7.3, we see that in terms of total fe and cpu, crs(P_{gs}, P_{lm}) is the best. crs(P_{gs+gl}, P_{lm}) and crs(P_{gsd}, P_{lm}) are runners-up, respectively, with respect to fe and cpu. In terms of success rate, sr, crs(P_{gs+gl}, P_{lm}) is the most reliable, followed by crs(P_{gs}, P_{lm}). If we compare the algorithms using all the problems used in our experiments, then crs(P_{gs+gl}, P_{lm}) would be the best both in reliability, in terms of success

Tab. 7.3: Comparison of the new crs algorithms using total results

Algorithm	fe	sr	cpu
crs(P_{gs}, P_{lm})	105902	3855	3.37
crs($P_{gs+gl}, \cdot$)	298360	3848	38.23
crs(P_{gs+gl}, P_{lm})	107561	3946	22.47
crs($P_{gs+gq}, \cdot$)	143512	3285	5.33
crs($P_{gq+gl}, \cdot$)	112065	3528	3.69
crs(P_{gsd}, P_{lm})	116105	3650	3.64

Tab. 7.4: Average performance of the new crs algorithms based on individual problems

	crs(P_{gs}, P_{lm})	crsP_{gs+gl}, P_{lm})	crs(P_{gsd}, P_{lm})
mfe_{rel}	22.3	19.7	11.8
σ_{rel}	32	47	40
sr	91.8	94	86.9

in locating the optimal value, and efficiency, in terms of the number of function evaluations. Notice also that the probabilistic algorithms crs($P_{gs+gl}, \cdot$), crs(P_{gs+gl}, P_{lm}) and crs(P_{gsd}, P_{lm}) solved more problems than crs(P_{gs}, P_{lm}), crs($P_{gs+gq}, \cdot$) and crs($P_{gq+gl}, \cdot$).

We now compare the new crs algorithms using mfe_{rel}, σ_{rel} and sr. However, unlike for the de algorithms, mfe_{rel} and σ_{rel} were calculated for only 42 problems. In Table 7.4, we present mfe_{rel}, σ_{rel} and sr of only three new crs algorithms. The crs($P_{gs+gl}, \cdot$) and crs($P_{gq+gl}, \cdot$) algorithms did not make any improvement in terms of fe on their respective counterparts, i.e. over crs($P_{gs}, \cdot$) and crs($P_{gq}, \cdot$), respectively. Therefore, we exclude them in our comparison using mfe_{rel}. Table 7.4 shows that in terms of mfe_{rel}, the average performance improvement of the algorithms crs(P_{gs}, P_{lm}), crs(P_{gs+gl}, P_{lm}) and crs(P_{gsd}, P_{lm}) is about 22%, 19.7% and 11.8%, respectively, while the corresponding standard deviations are about 32%, 47% and 40%, respectively. The high standard deviations here show that the performance improvement of the algorithms is problem dependent, i.e. percentage improvement widely varied from problem to problem.

Finally, we compare the new genetic algorithms using mfe_{rel}, σ_{rel} and sr. We calculated mfe_{rel} and σ_{rel} for only 40 problems, where all algorithms have positive success rate. The average fe_{rel} and standard deviation σ_{rel}, together with percentage success, sr, for these algorithms

are presented in Table 7.5. Table 7.5 shows that although the new ga algorithms made a significant improvement in terms of sr (compared to original ga($c_{r1}, \cdot$) with 75.8% sr), in terms of mfe_{rel}, the improvement is minimal. The standard deviations for the respective algorithms indicate that the performance improvement for all the problems is almost the same.

Tab. 7.5: Average performance of the new ga algorithms based on individual problems

	ga(c_{r1}, ℓ)	ga($c_{r2}, \cdot$)	ga(c_{r2}, ℓ)	ga($c_{r12}, \cdot$)	ga(c_{r12}, ℓ)
mfe_{rel}	0.6	2.2	7.3	5.0	9.9
σ_{rel}	22	26	25	22	21
sr	82.0	92.9	92.4	92.0	92.4

7.2 A comparison of ga(c_{r2}, ℓ), ga(c_{r12}, ℓ), crs(P_{gs+gl}, P_{lm}) and delb

In section 7.1, comparisons of the new de algorithms and the new crs algorithms were made. It was found that in terms of reliability, i.e. in terms of sr, crs(P_{gs+gl}, P_{lm}) is the best performer among the crs type algorithms. For the de type algorithms, it was found that delb is the best performer, in terms of sr. For the genetic algorithms, both ga($c_{r12}, \cdot$) and ga(c_{r12}, ℓ) were found to have about the same reliability. In this section, therefore, we compare these four algorithms based on the common test problems used in the numerical experiments. We compare the algorithms using the results of those functions for which all the methods succeeded in finding their global minimum. That is, we exclude the results of all algorithms for a problem if at least one of the algorithms has sr $= 0$. We present the summarized results in Table 7.6. A comparison of the algorithms shows that in terms of fe, crs(P_{gs+gl}, P_{lm}) is the best while delb is the worst. However, in terms of sr, delb is the best and crs(P_{gs+gl}, P_{lm}) is the runner-up.

We also compare the algorithms on the 10 dimensional problems that they all solved. For these problems, all the algorithms work with almost the same population size, i.e. $10n$ for delb, $10(n+1)$ for crs(P_{gs+gl}, P_{lm}) and 100 for the genetic algorithms. We would like to see how the algorithms compare when the population size is almost the same. We present the results for these 10 dimensional problems in Table 7.7. It is clear from Table 7.7 that fe for delb is the

Tab. 7.6: Comparison of ga(c_{r2}, ℓ), ga(c_{r12}, ℓ), crs(P_{gs+gl}, P_{lm}) and delb using total results

Algorithm	fe	sr
ga(c_{r2}, ℓ)	129831	3790
ga(c_{r12}, ℓ)	129600	3787
crs(P_{gs+gl}, P_{lm})	103293	3946
delb	714892	4058

Tab. 7.7: Comparison of ga(c_{r2}, ℓ), ga(c_{r12}, ℓ), crs(P_{gs+gl}, P_{lm}) and delb on 10 dimensional problems

Problem		ga(c_{r2}, ℓ)		ga(c_{r12},ℓ)		crs(P_{gs+gl}, P_{lm})		delb	
P	n	fe	sr	fe	sr	fe	sr	fe	sr
ACK	10	6784	98	6291	100	7999	100	20395	100
EXP	10	2530	100	2342	100	3133	100	7860	100
GW	10	3832	100	3659	100	4755	100	11841	100
LM2	10	3401	100	3165	100	4033	100	9222	100
ML	10	6894	2	8726	8	8979	37	173153	71
NF3	10	6931	100	7123	100	7280	100	79602	100
PP	10	3638	100	3600	100	4799	100	12107	100
RB	10	30667	93	31941	92	14661	100	192676	98
tr		64677	693	66847	700	55639	737	506856	769

highest for all problems. However, in terms of sr, delb is still the best performer with about 96.1% success rate (769 successes out of 800 runs), followed by crs(P_{gs+gl}, P_{lm}) with about 92.1%. The ga(c_{r2}, ℓ) algorithm comes last with about 86.6% success rate. This clearly shows that in terms of reliability, delb is the best. In terms of efficiency, crs(P_{gs+gl}, P_{lm}) outperforms the other algorithms.

Chapter 8

Conclusion

The work in this thesis was devoted to the improvements of some population set-based methods for unconstrained global optimization with continuous variables. The focus of the research was mainly on the differential evolution algorithm, the controlled random search algorithm and the real coded genetic algorithm with arithmetic crossover. We presented a number of hybrid methods involving these algorithms. We introduced some new ideas, e.g. some local search techniques in these algorithms. We also introduced a number of new point generation schemes and a scheme that probabilistically combine point generation schemes in an algorithms. We combined the existing point generation schemes of these algorithms with the new ones in designing new algorithms.

In the new methods based on differential evolution, we first introduced some localizations in the mutation and acceptance phases of the de algorithm. We then hybridized the differential evolution using the point generation schemes of the crs2 algorithms and the electromagnetism-like algorithm. The electromagnetism-like algorithm is also a population set-based method that was recently proposed for nondifferentiable and nonconvex global optimization problems. It uses the concept of attraction-repulsion of forces in generating points.

For the new methods based on the crs2 algorithm, we hybridized the original crs2 algorithm by combining the point generation scheme of the crs2 algorithm with the generation scheme of other methods. In particular, for some algorithms, we combined the crs2 point generation scheme with that of the de algorithm and for some, we combined the crs2 point generation scheme with that of a trust region method. As in the new de algorithms, we also incorporated

some localizations in the crs algorithms for exploration of searches in the earlier stages of the search and for faster convergence at the later stages.

Lastly, we hybridized a real coded genetic algorithm that uses the arithmetic crossover rule for the generation of children. We suggested a crossover rule, called integrated crossover rule, that generates one child using a de mutation rule and another child using a 'simplex-like' crossover rule. We then combined the arithmetic crossover rule with this rule to have a scheme that generates two children either with the arithmetic crossover or the integrated crossover rule.

Whenever different point generation schemes were combined, we used a probabilistic adaptation scheme to probabilistically combine them. This probabilistic adaptation scheme rewarded a scheme for meeting the intended target and penalized otherwise.

The efficiency and reliability of the new methods were evaluated using 50 test problems from the literature. The efficiency of each method was evaluated using the number of function evaluations and cpu times the method needed to locate the optimal value. The reliability of each method was judged using the success rate of the method in locating the optimal value. The results showed that the new methods are more efficient, in terms of the number of function evaluations and cpu times, and more reliable, in terms of locating the optimal values, than their original counterparts. However, it was also evident from the results that not all combined point generation schemes always held advantages over the original schemes.

Convergence of purely heuristic methods depends on the effort, in terms of the number of function evaluations and cpu time, the methods need to solve problems. Thus, heuristic methods prove their usefulness by numerical testing. Results showed that the new methods needed less number of function evaluations and cpu time to reach the global minimum. Thus, the new methods converge faster than their original counterparts.

Hybridization has been one way of trying to come up with methods that are applicable to a wide range of problems. Through hybridization, a lot of methods that are more reliable can be developed. Therefore, more research is still needed in finding even more efficient global optimization methods that are applicable to a wide range of complex optimization problems.

Chapter 9

Author's contribution

The ideas reported in this study are mainly the author's ideas. The author also wrote the papers. In all the papers, the co-author (supervisor), Prof. Montaz Ali, provided some general ideas that helped refine the papers in both scientific and linguistic aspects. The co-author also provided some general ideas to the work, but the author of this thesis made all the implementations of the algorithms (except for the crs(P_{gs}, $P_{l\beta}$) (crs4) algorithm) and carried out all the experiments by himself. The Pascal code for crs4 was provided by the supervisor. The supervisor also closely supervised the whole thesis. The supervision included exchange of ideas and proof reading of the manuscripts. The supervisor freely provided his professional expertise and knowledge.

Bibliography

[1] Aarts, E. and Korst, J. *Simulated Annealing and Boltzmann Machines*. Wiley, Chichester, 1989.

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

[3] Ali, M.M., Khompatraporn, C. and Zabinsky, Z.B. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization* (in press).

[4] Ali, M.M. and Storey, C. Modified controlled random search algorithms. *Intern. Journal of Computer Mathematics 53* (1994), 229–235.

[5] Ali, M.M. and Storey, C. Aspiration based simulated annealing algorithm. *Journal of Global Optimization 11* (1997), 181–191.

[6] Ali, M.M., Törn, A. and Viitanen, S. A direct search variant of the simulated annealing algorithm for optimization involving continuous variables. *Computers & Operations Research 29* (2002), 87–102.

[7] Ali, M.M., T̈orn, A. and Viitanen, S. A numerical comparison of some modified controlled random search algorithms. *Journal of Global Optimization 11* (1997), 377–385.

[8] Ali, M.M. and Törn, A. Population set-based global optimization algorithms: some modifications and numerical studies. *Computers & Operations Research 31*, 10 (2004), 1703–1725.

[9] Ali, M.M. and Törn, A. Topographical differential evolution using pre-calculated differentials. In *Stochastic and Global Optimization*, G. Dzemyda, V. Saltenis and A. Zilinskas, Eds. Kluwer Academic Publishers, Dordrecht, 2002, pp. 1–17.

[10] Babu, B.V. and Sastry, K.K.N. Estimation of heat transfer parameters in a trickle-bed reactor using differential evolution and orthogonal collocation. *Computers and Chemical Engineering 23*, 3 (1999), 327–339.

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

[12] Bazaraa, M.S., Sherali, H. and Shetty, C.M. *Nonlinear Programming: Theory and Algorithms*, 2nd ed. John Wiley, New York, 1993.

[13] Bergey, P.K. and Ragsdale, C. Modified differential evolution: a greedy random strategy for genetic recombination. *Omega 33,* 3 (2005), 255–265.

[14] van den Bergh, F. and Engelbrecht, A.P. A study of particle swarm optimization particle trajectories. *Information Sciences* (in press).

[15] Birbil S.I. and Fang S.-C. An electromagnetism-like mechanism for global optimization. *Journal of Global Optimization 25,* 3 (2003), 263–282.

[16] Birbil S.I., Fang S.-C. and Sheu R.-L. On the convergence of a population-based global optimization. *Journal of Global Optimization 30* (2004), 301–318.

[17] Björkman, M. and Holmström, K. Global optimization using the DIRECT algorithm in matlab. *Advanced Modeling and Optimization 1,* 2 (1999), 17–37.

[18] Brachetti, P., De Felice Ciccoli, M., Di Pillo, G. and Lucidi, S. A new version of the Price's algorithm for global optimization. *Journal of Global Optimization 10* (1997), 165–184.

[19] Bressloff, P.C. Stochastic dynamics of reinforcement learning. *Network: Computation in Neural Systems 6* (1995), 289–307.

[20] Chatterjee, A. and Siarry, P. Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. *Computers & Operations Research* (in press).

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

[22] Chelouah, R. and Siarry, P. Tabu search applied to global optimization. *European Journal of Operational Research 123,* 2 (2000), 256–270.

[23] Chelouah, R. and Siarry, P. A hybrid method combining continuous tabu search and Nelder-Mead simplex algorithms for the global optimization of multiminima functions. *European Journal of Operational Research 161,* 3 (2005), 636–654.

[24] Chelouah, R. and Siarry, P. Genetic and Nelder-Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. *European Journal of Operational Research 148,* 2 (2003), 335–348.

[25] Chong, E.K.P. and $\dot{Z}$ak, S.H. *An Introduction to Optimization*, 2nd ed. John Wiley & Sons, New York, USA, 2001.

[26] Corne, D., Dorigo, M. and Glover, F., Eds. *New Ideas in Optimization.* McGraw-Hill, Cambridge, 1999.

[27] Deb, K. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, New York, 2001.

[28] Debels D, De Reyck B, Leus R, Vanhoucke M. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *European Journal of operational research* (in press).

[29] Dekkers, A. and Aarts, E. Global optimization and simulated annealing. *Mathematical Programming 50* (1991), 367–393.

[30] Dorigo, M. and Di Caro, G. The ant colony optimization metaheuristic. In *New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover, Eds. McGraw-Hill, Cambridge, 1999, pp. 11–32.

[31] Dorigo, M. and Stützle, T. *Ant Colony Optimization*. MIT Press, Cambridge, 2004.

[32] Elegbede, C. Structural reliability assessment based on particle swarm optimization. *Structural Safety 27*, 2 (2005), 171–186.

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

[34] Floudas, A. and Pardalos, M. *A Collection of Test Problems for Constrained Global Optimization Algorithms*. Springer-Verlag, Berlin, 1990.

[35] Floudas, A. and Pardalos, M., Eds. *Recent Advances in Global Optimization*. Princeton University Press, Princeton, 1992.

[36] Gendreau, M. Recent advances in tabu search. In *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, Eds. Kluwer Academic Publishers, Boston, 2002, pp. 369–377.

[37] Glover, F. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research 13* (1986), 533–548.

[38] Glover, F. Tabu search-Part I. *ORSA Journal of Computing 1*, 3 (1989), 190–206.

[39] Glover, F. Tabu search: a tutorial. *Interfaces 20*, 4 (1990), 74–94.

[40] Glover, F. and Kochenberger, G.A., Eds. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Boston, 2003.

[41] Glover, F. and Laguna, M. *Tabu Search*. Kluwer Academic Publishers, Dordrecht, 1997.

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

[43] Gravel, M., Price, W.L. and Gagne, C. Scheduling continuous casting of aluminium using a multiple objective ant colony optimization metaheuristic. *European Journal of Operational Reseach 143* (2002) 218–229.

[44] Haslinger, J., Jedelský, D., Kozubek, T. and Tvrdík, J. Genetic and random search methods in optimal shape design problems. *Journal of Global Optimization 16* (2000) 109–131.

[45] Hedar, A.-L. and Fukushima, M. Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization. *Optimization Methods and Software 17*, 5 (2002), 891–912.

[46] Herrera, F. and Lozano, M. Two-loop real-coded genetic algorithms with adaptive control of mutation step sizes. *Applied Intelligence 13* (2000), 187–204.

[47] Herrera, F., Lozano, M. and Molina, D. Continuous scatter search: an analysis of the integration of some combination methods and improvement strategies. *European Journal of Operational Research* (in press).

[48] Herrera, F., Lozano, M. and Sanchez, A.M. A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study. *International Journal of Intelligent Systems 18*, 3 (2003) 309–338.

[49] Holland, J.H. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, 1975.

[50] Horst, R. and Pardalos, P.M., Eds. *Handbook of Global Optimization*. Kluwer Academic Publishers, Dordrecht, 1995.

[51] Horst, R. and Tuy, H. *Global Optimization: Deterministic Approaches*, 3rd ed. Springer-Verlag, Berlin, 1996.

[52] Hrstka, O. and Kučerová, A. Improvements of real coded genetic algorithms based on differential operators preventing premature convergence. *Advances in Engineering Software 35*, 3-4 (2005), 237–246.

[53] Hrstka, O., Kučerová, A., Lepš, M. and Zeman, J. A competitive comparison of different types of evolutionary algorithms. *Computers and Structures 81* (2003), 1979–1990.

[54] Hu, Y.F., Maguire, K.C., Cokljat, D. and Blake, R.J. Parallel controlled random search algorithms for shape optimization. In *Parallel Computational Fluid Dynamics*, D.R. Emerson, A. Ecer, J. Periaux, N. Satofuka, Eds. Amsterdam, 1997, pp. 345–352.

[55] Ingber, L. Simulated annealing: practice versus theory. *Mathematical and Computer Modelling 18*, 11 (1993), 29–57.

[56] Jones, D.R., Perttunen, C.D. and Stuckman, B.E. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications 79*, 1 (1993), 157–181.

[57] Kalganova, T. and Strechen, N. Genetic algorithm approach to find the best input variable partitioning. In *Proceedings of the 3rd Nordic Workshop on GA*. Helsinki, 1997, pp. 101–290.

[58] Kearfott, R.B. *Rigorous Global Search: Continuous Problems*. Kluwer Academic Publishers, Dordrecht, 1996.

[59] Kelly, C.T. *Iterative Methods for Optimization*. SIAM, Philadelphia, 1999.

[60] Kirkpatrick, S., Gelatt, C.D. and Vecci, M.P. Optimization by simulated annealing. *Science 220* (1983), 671–680.

[61] Kovacevic-Vujcic, V.V., Cangalovic, M.M., Ašic, M.D., Ivanovic, L. and Drazic, M. Tabu search methodology in global optimization. *Computers and Mathematics with Applications 37* (1999), 125–133.

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

[63] Laguna, M. Scatter search. In *Handbook of Applied Optimization*, P.M. Pardalos and M.G.C. Resende, Eds. Oxford University Press, Oxford, 2002, pp 183–193.

[64] Lampinen, J. and Ivan, Z. Mechanical engineering design optimization by differential evolution. In *New ideas in optimization*, D. Corne, M. Dorigo, F. Glover, Eds. McGraw-Hill, Cambridge, 1999, pp. 127–46.

[65] Laskari, E.C., Parsopoulos, K.E. and Vrahatis, M.N. Evolutionary operators in global optimization with dynamic search trajectories. *Numerical Algorithms 34* (2003), 393–403.

[66] Li, J.R., Khoo, L.P. and Tor, S.B. A tabu-enhanced genetic algorithm approach for assembly process planning. *Journal of Intelligent Manufacturing 14* (2003), 197–208.

[67] Maniezzo, V. and Carbonaro, A. Ant colony optimization: an overview. In *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, Eds. Kluwer Academic Publishers, Boston, 2002, pp. 469–492.

[68] Marazzi, M. and Nocedal, J. Wedge trust region methods for derivative free optimization. *Mathematical Programming 91* (2002), 289–305.

[69] Michalewicz, Z. Evolutionary computations and heuristics. In *Meta-Heuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, Eds. Kluwer Academic Publishers, Boston, 1996, pp. 37–52.

[70] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, 1996.

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

[72] Miettinen, K., Mäkelä, M.M. and Maaranen, H. Efficient hybrid methods for global continuous based on simulated annealing. *Computers & Operations Research* (in press).

[73] Mohan, C. and Shanker, K. A controlled random search technique for global optimization using quadratic approximation. *Asia-Pacific Journal of Operational Research 11* (1994), 93–101.

[74] Moloi, N.P. and Ali, M.M. An iterative global optimization algorithm for potential energy minimization. *Computational Optimization and Applications 30*, 2 (2005) 119–132.

[75] Mühlenbein, H. and Schlierkamp-Voosen, D. Predictive models for the breeder genetic algorithms I: continuous parameter optimization. *Evolutionary Computation 1* (1993), 25–49.

[76] Najim, K., Pibouleau, L. and Le Lann, M.V. Optimization technique based on learning automata. *Journal of Optimization Theory and Applications 64* (1990), 331–347.

[77] Nelder, J.A. and Mead, R. A simplex method for function minimization. *Computer Journal 7* (1965), 308–313.

[78] Nocedal, J. and Wright, S.J. *Numerical Optimization*. Springer Series in Operations Research. Springer, New York, 1999.

[79] Okamoto, M, Nonaka, T., Ochiai, S and Tominaga, D. Nonlinear numerical optimization with use of a hybrid genetic algorithm incorporating the modified Powell method. *Applied Mathematics and Computation 91* (1998), 63–72.

[80] Onwubolu, G.D. and Babu, B.V. New Optimization Techniques in Engineering. *Studies in Fuzziness and Soft Computing 141*. Springer-Verlag, New York, 2004.

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

[82] Özdamar, L. and Demirhan, M. Experiments with new stochastic global optimization search techniques. *Computers & Operations Research 27*, 9 (2002), 841–865.

[83] Pardalos, P.M. and Resende, M.G.C. *Handbook of Applied Optimization*. Oxford University Press, Oxford, 2002.

[84] Pardalos, P.M. and Romeijn, H.E., Eds. *Handbook of Global Optimization 2*. Kluwer Academic Publishers, Boston, 2002.

[85] Price K. An introduction to differential evolution. In *New Ideas in Optimization*, D. Corne, M. Dorigo, F. Glover, Eds. McGraw-Hill, Cambridge, 1999, pp. 79–108.

[86] Price, W.L. A controlled random search procedure for global optimization. *The Computer Journal 20* (1978), 367–370.

[87] Price, W.L. Global optimization algorithms for a CAD workstation. *Journal of Optimization Theory and Applications 55*, 1 (1987), 133–146.

[88] Price, W.L. Global optimization by controlled random search. *Journal of Optimization Theory and Applications 40*, 3 (1983), 333–348.

[89] Reeves, C. Genetic algorithms. In *Handbook of Metaheuristics*, F. Glover and G.A. Kochenberger, Eds. Kluwer Academic Publishers, Boston, 2003, pp. 55–82.

[90] Reeves, C.R. *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill, London, 1995.

[91] Resende, M.G.C. and de Sousa, J.P., Eds. *Metaheuristics: Computer Decision-Making*. Kluwer Academic Publishers, Dordrecht, 2003.

[92] Ribeiro, C.C. and Hansen, P., Eds. *Essays and Surveys in Metaheuristics*. Kluwer Academic Publishers, Boston, 2002.

[93] Rinnooy Kan, A.H.G. and Timmer G.T. Stochastic global optimization methods, Part I: clustering methods. *Mathematical Programming 39* (1987), 27–56.

[94] Rinnooy Kan, A.H.G. and Timmer G.T. Stochastic global optimization methods, Part II: multi level methods. *Mathematical Programming 39* (1987), 57–78.

[95] Sarimveis, H. and Nikolakopoulos, A. A line up evolutionary algorithm for solving nonlinear constrained optimization problems. *Computers & Operations Research 32* (2005), 1499–1514.

[96] Shubert, B. A sequential method seeking the global minimum of a function. *SIAM Journal on Numerical Analysis 9* (1972), 379–388.

[97] Snyman, J.A. and Fatti, L.P. A multi-start global minimization algorithm with dynamic search trajectories. *Journal of Optimization Theory and Applications 54*, 1 (1987), 121–141.

[98] Storn, R. System design by constraint adaptation and differential evolution. *IEEE Transactions on Evolutionary Computation 3*, 1 (1999), 22–34.

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

[100] Talbi, E.-G. Taxonomy of hybrid metaheuristics. *Journal of Heuristics 8*, 5 (2002), 541–564.

[101] T*ö*rn, A., Ali, M.M. and Viitanen, S. Stochastic global optimization: problem classes and solution techniques. *Journal of Global Optimization 14* (1999), 437–447.

[102] Törn, A. and Žilinskas, A. Global Optimization. *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1989.

[103] Trafalis, T.B. and Kasap, S. A novel metaheuristics approach for continuous optimization. *Journal of Global Optimization 23* (2002), 171–190.

[104] Trelea, I.C. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters 85*, 6 (2003), 317–325.

[105] Tsutsui, S. and Goldberg, D.E. Search space boundary extension method in real-coded genetic algorithms. *Information Sciences 133* (2001), 229–247.

[106] Wang, P.P. and Chen, D.-S. Continuous optimization by a variant of simulated annealing. *Computational Optimization and Applications 6*, 1 (1996), 59–71.

[107] Voss, S., Martello, S., Osman, I.H. and Roucairol, C., Eds. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Boston, 1999.

[108] Yang, R. and Douglas, I. Simple genetic algorithm with local tuning: efficient global optimization technique. *Journal of Optimization Theory and Applications 2* (1998), 449–465.

[109] Yun, Y., Gen, M. and Seo, S. Various hybrid methods based on genetic algorithm with fuzzy logic controller. *Journal of Intelligent Manufacturing 14* (2003), 401–419.

[110] Zabinsky, Z.B. *Stochastic Adaptive Search for Global Optimization*. Kluwer Academic Publishers, London, 2003.

[111] Zabinsky, Z.B. Stochastic methods for practical global optimization. *Journal of Global Optimization 13* (1999), 433–444.

Appendix A

More detailed description of DIRECT algorithm

We briefly present the DIRECT algorithm, developed by Jones et.al. [56]. The DIRECT algorithm is a modification of the original Lipschitzian approach that eliminates the need to provide a Lipschitz constant. It was designed to overcome some of the problems that Lipschitzian optimization encounters. We begin by giving a brief discussion of Lipschitzian optimization.

A.1 Lipschitzian optimization

Consider a problem of finding the global minimum of a function $f : \Omega \to R$ defined in a closed interval $\Omega = [l, u]$.

Definition 1 *The function f is called Lipschitz continuous on Ω with Lipschitz constant K if*

$$\mid f(x) - f(x') \mid \leq K \mid x - x' \mid \ \forall x, x' \in \Omega. \tag{A.1}$$

If f is Lipschitz continuous with constant K, then this information can be used to construct an iterative algorithm to find the minimum of f. The Shubert's algorithm [96] is a good example of an application of this idea. If we substitute l and u for x' in (A.1), then f must satisfy the two inequalities

$$f(x) \geq \quad f(l) - K(x - l), \tag{A.2}$$

$$f(b) \geq \quad f(u) + K(x - u) \tag{A.3}$$

for any $x \in \Omega$. The lines that correspond to these two inequalities form a V-shape below f, as shown in figure A.1. The point of intersection for the two lines provides the first estimate of the minimum of f. If we denote this point by x_1 and the corresponding lower bound of f by y_1, then we have

$$x_1 = \quad (l+u)/2 + [f(l) - f(u)]/(2K) \tag{A.4}$$

$$y_1 = \quad [f(l) + f(u)]/2 - K(u-l). \tag{A.5}$$

These two equations form the core of the Shubert's algorithm. The point x_1 divides the search space into two search regions, $[l, x_1]$ and $[x_1, u]$. Shubert's algorithm continues by performing the same operation on the regions $[l, x_1]$ and $[x_1, u]$, dividing the region with the lower function value.

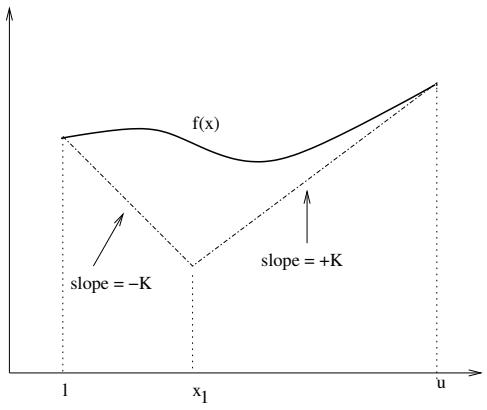

Fig. A.1: Computing the Lipschitzian lower bound for an interval

There are several problems with this type of algorithm. The first problem lies in that, since the Lipschitz constant must be an upper bound on the rate of change of the function, it leads to slow convergence of the Shubert's algorithm if the constant K is too high. On the other hand, if K is too low then the result obtained may not be a minimum of f. The second shortcoming of Lipschitzian optimization is its computational complexity in higher dimensions. Specifying the Lipschitzian constant is also a problem because, practically, it may not exist or may not be easily computed.

A.2 DIRECT algorithm

The DIRECT algorithm was developed to eliminate the shortcomings of Lipschitzian optimization discussed in the previous section. Instead of evaluating the function at the endpoints of an interval, the DIRECT algorithm evaluates it at the centre point of the search space. This means that the algorithm can be initialized by sampling only the centre point of the search space as opposed to the vertices of the search space.

The DIRECT algorithm works in a normalized space $\bar{\Omega} = \{x \in R^n : 0 \leq x_i \leq 1\}$, referring to the original space only when evaluating the function f. If the centre of the space is c_1, then the algorithm begins by finding $f(c_1)$. The next step is to divide the hypercube $\bar{\Omega}$. This is done by evaluating the function at the points $c_1 \pm \delta \mathbf{e}_i,\ , i = 1, 2, \cdots, n$, where δ is one third of the side length of the hypercube, and $\mathbf{e}_i$ is the ith unit vector. The hypercube is divided as follows. Let

$$w_i = \min\{f(c_1 + \delta \mathbf{e}_i), f(c_1 - \delta \mathbf{e}_i)\} \tag{A.6}$$

be the best of the function values sampled along the ith dimension. Divide the dimension with the smallest w_i into thirds, so that $c_1 \pm \delta \mathbf{e}_i$ are the centres of the new hyper-rectangles. Then split the rectangle containing c_1 into thirds along the dimension with the next smallest w_i. The procedure for dividing rectangles consists of three basic steps. These are described below.

Procedure for dividing rectangles

Step 1 Identify the set I of dimensions with the maximum side length. Let δ be one-third of this maximum side length.

Step 2 Evaluate the function at the points $c_1 \pm \delta \mathbf{e}_i$ for all $i \in I$.

Step 3 Divide the rectangle containing c_1 into thirds along the dimensions in I, starting with the dimension with the lowest value of

$$w_i = \min\{f(c_1 + \delta \mathbf{e}_i), f(c_1 - \delta \mathbf{e}_i)\},$$

and continuing to the dimension with the highest w_i.

DIRECT searches globally and locally, to determine which rectangles are potentially optimal, by dividing all hyper-rectangles that meet the criteria of Definition 2

Definition 2 *Suppose that we have a partition of the unit hypercube into m hyper-rectangles. Let $\epsilon > 0$ be a positive constant, and let $f_{\min}$ be the current best function value. A hyper-rectangle j is said to be potentially optimal if there exists some $\bar{K} > 0$ such that*

$$f(c_j) - \bar{K} d_j \leq f(c_i) - \bar{K} d_i \,, \qquad \forall i = 1, 2, \cdots, m, \tag{A.7}$$

$$f(c_j) - \bar{K} d_j \leq f_{\min} - \epsilon \mid f_{\min} \mid . \tag{A.8}$$

In the definition above, c_j is the centre of the jth hyper-rectangle, and d_j is a measure for this hyper-rectangle. In Jones et.al. [56], this measure is chosen as the distance from c_j to the vertices.

The process of identifying the potentially optimal hyper-rectangles and subdividing them as described continues until a prescribed stopping condition is met. Below we present the DIRECT algorithm [56].

DIRECT Algorithm

Step 1 Normalize the search region to be a unit hypercube with centre c_1.

Step 2 Evaluate $f(c_1)$. Set $f_{\min} = f(c_1)$, $m = 1$, $t = 1$ (iteration counter).

Step 3 Identify the set S of potentially optimal rectangles.

Step 4 Select any rectangle $j \in S$.

Step 5 Using the procedure described earlier, determine where to sample in rectangle j and how to divide the rectangle into sub rectangles. Update $f_{\min}$ and set $m = m + \Delta m$, where Δm is the number of new sampled points.

Step 6 Set $S = S - j$. if $S \neq \emptyset$ go to Step 4.

Step 7 Check stopping criterion. If stopping criterion not satisfied, go to Step 3.

Appendix B

Description of a real coded genetic algorithm

Real coded genetic algorithms (ga) have been used in recent years to solve continuous optimization problems [69, 70]. Real coded genetic algorithms are based on the same concept as a typical genetic algorithm. The difference between real coded genetic algorithms and the typical ga is that points in the search space are represented as real numbers instead of binary strings. Real coded genetic algorithms are also easier to hybridize with other optimization methods and are faster than the traditional binary coded genetic algorithms in terms of convergence [69, 70, 109]. They are also applicable to a wide range of problems with high dimensions.

The difference in representation of values leads to the need to introduce new recombination (crossover) and mutation operators in a real coded genetic algorithm. In the following sections, we give details of some of the recombination and mutation operators that have been proposed in the literature for real coded genetic algorithms. We define parent strings as $x = (x_1, x_2, \cdots, x_n)$ and $y = (y_1, y_2, \cdots, y_n)$, and an offspring string as $z = (z_1, z_2, \cdots, z_n)$, where all $x_i, y_i, z_i \in R$, $i = 1, 2, \cdots, n$ and n is the dimension of the problem being solved.

B.1 Recombination

There are different types of operators that have been suggested in the literature for recombination. Some of the operators suggested are:

- *Discrete recombination*. In discrete recombination, the ith component of an offspring z is given as:

$$z_i = x_i \text{ or } y_i,$$

where x_i or y_i is chosen with probability 0.5. Thus, the point z is a mixture of the components of the parents points.

- *Extended line recombination*. In extended line recombination, the ith component of an offspring z is given as:

$$z_i = x_i + \alpha(y_i - x_i),$$

where α is chosen randomly in the range $[-\delta, 1+\delta]$. This recombination generates points on the line between the parents. Typically, $\delta \in (0, 1)$.

- *Extended intermediate recombination*. In this recombination, the ith component of z is given by

$$z_i = x_i + \alpha_i(y_i - x_i),$$

where $\alpha_i \in [-\delta, 1+\delta]$ is chosen randomly. In this recombination, points are generated inside the hypercube defined by the parents, x and y.

- *Heuristic recombination*. In heuristic recombination, the ith component of z is given as

$$z_i = x_i + r(x_i - y_i),$$

where $r \in [0, 1]$ is a random number and the parent x is better than the parent y. Thus, in heuristic recombination, the offspring z is generated as a coordinate-wise reflection of the worse parent y (in terms of objective function value) through the better parent x.

B.2 Mutation

In mutation, the new offspring z may be modified. Each component of the real coded string is considered and modification takes place if a randomly generated number is less than a pre-defined mutation rate. Thus, each variable z_i is selected for mutation with some probability. There are different types of mutation operators that have been proposed, for example,

- *Discrete mutation*. In discrete mutation, a component z_i is modified as

$$z_i = z_i \pm \delta_i \sum_{j=1}^{k-1} \alpha_j \cdot 2^{-j},$$

where $\alpha_j \in \{0, 1\}$ are chosen to be equal to 1 with probability $1/k$, k is an integer, called precision constant, and $\delta_i = 0.1(u_i - l_i)$, where l_i and u_i are the ith lower and upper bounds of z_i, respectively.

- *Continuous mutation*. In this mutation scheme, z_i is modified as

$$z_i = z_i \pm \delta_i \cdot 2^{-k\alpha},$$

where α is chosen uniformly from the range $[0, 1]$ and k is the precision constant.

- *Non uniform mutation*. In non uniform mutation, an element z_i is modified as

$$z_i = \begin{cases} z_i + \triangle(t, u_i - z_i) & \text{if} \quad \tau = 0 \\ z_i - \triangle(t, z_i - l_i) & \text{if} \quad \tau = 1, \end{cases}$$

where τ is a random digit that takes either the value zero or one, and

$$\triangle(t, y) = y \left(1 - r^{(1-\frac{t}{T})^b}\right),$$

where r is a random number from $[0, 1]$, T is the maximal generation number, t is the current iteration number, and b is a parameter provided by the user which determines the degree of non-uniformity.

Appendix C

Standard differential evolution (de) algorithm

The differential evolution algorithm utilizes N, n-dimensional parameter vectors $x_{i,k}$, $i = 1, 2, \cdots, N$, as a population to search the feasible region Ω. The index k denotes the iteration (or generation) number of the algorithm. The initial population,

$$S = \{x_{1,0}, x_{2,0}, \cdots, x_{N,0}\}, \tag{C.1}$$

where $k = 0$, is taken to be uniformly distributed in the search region. At each iteration, all vectors in S are targeted for replacement. Therefore, N competitions are held to determine the members of S for the next iterations. This is achieved by using the mutation, crossover and acceptance. This process (mutation, crossover and acceptance) is repeated until a pre-defined stopping condition is met.

C.1 Mutation

For each target vector $x_{i,k}$, $i = 1, 2, \cdots, N$, a mutant vector $\hat{x}_{i,k}$ is obtained by

$$\hat{x}_{i,k} = x_{\alpha,k} + F(x_{\beta,k} - x_{\gamma,k}), \tag{C.2}$$

where $\alpha, \beta, \gamma \in \{1, 2, \cdots, N\}$ are mutually distinct random indices and are also different from the current target index i. The vector $x_{\alpha,k}$ is known as the base vector and $F > 0$ is a scaling parameter. If the mutant vector $\hat{x}_{i,k} \notin \Omega$ then the mutation operation is repeated.

C.2 Crossover

The crossover operator is applied to obtain a trial vector $y_{i,k}$ from $\hat{x}_{i,k}$ and $x_{i,k}$. The crossover is defined by

$$y_{i,k}^{j} = \begin{cases} \hat{x}_{i,k}^{j} & \text{if } R^j \leq C_R \text{ or } j = I_i \\ x_{i,k}^{j} & \text{if } R^j > C_R \text{ and } j \neq I_i \ , \end{cases} \tag{C.3}$$

where I_i is a randomly chosen integer in the set I, i.e., $I_i \in I = \{1, 2, \cdots, n\}$; the superscript j represents the j-th component of respective vectors; $R^j \in (0, 1)$, drawn randomly for each j.

C.3 Acceptance

After all N trial vectors $y_{i,k}$ have been generated, acceptance is applied. In the acceptance phase, the function value at the trial vector, $f(y_{i,k})$, is compared to $f(x_{i,k})$, the value at the target vector and the target vector is updated using

$$x_{i,k+1} = \begin{cases} y_{i,k} & \text{if } f(y_{i,k}) \leq f(x_{i,k}) \\ x_{i,k} & \text{otherwise.} \end{cases} \tag{C.4}$$

Appendix D

Controlled random search (crs2)

The controlled random search (crs2) algorithm, like de and ga, uses a population set S of points to search the space Ω. However, unlike in de and ga, crs2 gradually contracts S by replacing the current worst point in S with a better trial point through repeated use of simplexes. At each iteration, therefore, the current worst point x_h in S is targeted. A trial point is generated by forming a simplex. The simplex is formed using $n+1$ distinct points, $x_{p(1)}, x_{p(2)}, \cdots, x_{p(n+1)}$, where $x_{p(2)}, x_{p(3)}, \cdots, x_{p(n+1)}$ are chosen at random from S and the point $x_{p(1)}$ is always the current best point x_l in S. The trial point is obtained by reflecting the point $x_{p(n+1)}$ in the centroid of the remaining n points of the simplex, as in the Nelder and Mead algorithm [77]. The trial point $\tilde{x}$ is thus given as

$$\tilde{x} = 2G - x_{p(n+1)}, \tag{D.1}$$

where the centroid G is given by

$$G = \frac{1}{n} \sum_{j=1}^{n} x_{p(j)}. \tag{D.2}$$

If the trial point $\tilde{x} \notin \Omega$, then another trial point is found using another set of $n+1$ distinct points. The crs2 algorithm is similar to crs1, described in detail in paper **B]**, except that in crs2 the current best point x_l is used in forming the simplex.

Appendix E

A collection of benchmark global optimization test problems

In this appendix, we present 52 popular test problems which are often used by global optimization researchers. Some of these problems can be found in textbooks, in individual research articles, or at different web sites. A collection of these problems can also be found in Ali et. al. [3]. The 51st problem was taken from Chelouah and Siarry [24]. Please note that in several cases the optimal point vector and corresponding global point are known only as a numerical approximation.

1. Ackley's Problem (ACK)

$$\min_x f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}{x_i}^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \quad \text{(E.1)}$$

$$\text{subject to} \quad -30 \leq x_i \leq 30, i \in \{1, 2, \ldots, n\}. \quad \text{(E.2)}$$

The number of local minima is not known. The global minimum is located at the origin with $f(x^*) = 0$. Tests were performed for $n = 10, 20$.

2. Aluffi-Pentini's Problem (AP)

$$\min_x f(x) = 0.25{x_1}^4 - 0.5{x_1}^2 + 0.1x_1 + 0.5{x_2}^2 \quad \text{(E.3)}$$

$$\text{subject to} \quad -10 \leq x_1, x_2 \leq 10. \quad \text{(E.4)}$$

The function has two local minima, one of them is global with $f(x^*) \approx -0.3523$ located at $(-1.0465, 0)$.

3. Becker and Lago Problem (BL)

$$\min_x f(x) = \quad (|x_1| - 5)^2 + (|x_2| - 5)^2 \tag{E.5}$$

$$\text{subject to} \qquad -10 \leq x_1, x_2 \leq 10. \tag{E.6}$$

The function has four minima located at $(\pm 5, \pm 5)$, all with $f(x^*) = 0$.

4. Bohachevsky 1 Problem (BF1)

$$\min_x f(x) = \quad x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7 \tag{E.7}$$

$$\text{subject to} \qquad -50 \leq x_1, x_2 \leq 50. \tag{E.8}$$

The number of local minima is unknown but the global minimizer is located at $x^* = (0, 0)$ with $f(x^*) = 0$.

5. Bohachevsky 2 Problem (BF2)

$$\min_x f(x) = \quad x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3 \tag{E.9}$$

$$\text{subject to} \qquad -50 \leq x_1, x_2 \leq 50. \tag{E.10}$$

The number of local minima is unknown but the global minimizer is located at $x^* = (0, 0)$ with $f(x^*) = 0$.

6. Branin Problem (BP)

$$\min_x f(x) = \quad a(x_2 - b{x_1}^2 + cx_1 - d)^2 + g(1 - h)\cos(x_1) + g\,, \tag{E.11}$$

$$\text{subject to} \qquad -5 \leq x_1 \leq 10, 0 \leq x_2 \leq 15, \tag{E.12}$$

where $a = 1$, $b = 5.1/(4\pi^2)$, $c = 5/\pi$, $d = 6$, $g = 10$, $h = 1/(8\pi)$. There are three minima, all global, in this region. The minimizers are

$$x^* \approx (-\pi, 12.275), (\pi, 2.275), (3\pi, 2.475)$$

with $f(x^*) = 5/(4\pi)$.

7. Camel Back–3 Three Hump Problem (CB3)

$$\min_x f(x) = \quad 2x_1^2 - 1.05x_1^4 + \tfrac{1}{6}x_1^6 + x_1x_2 + x_2^2 \tag{E.13}$$

$$\text{subject to} \qquad -5 \leq x_1, x_2 \leq 5. \tag{E.14}$$

The function has three local minima, one of them is global located at $x^* = (0, 0)$ with $f(x^*) = 0$.

8. Camel Back–6 Six Hump Problem (CB6)

$$\min_x f(x) = \quad 4x_1^2 - 2.1x_1^4 + \tfrac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 \tag{E.15}$$

$$\text{subject to} \qquad -5 \leq x_1, x_2 \leq 5. \tag{E.16}$$

This function is symmetric about the origin and has three conjugate pairs of local minima with values $f \approx -1.0316, -0.2154, 2.1042$. The function has two global minima at $x^* \approx (0.089842, -0.712656)$ and $(-0.089842, 0.712656)$ with $f(x^*) \approx -1.0316$.

9. Cosine Mixture Problem (CM)

$$\max_x f(x) = \qquad 0.1\sum_{i=1}^{n}\cos(5\pi x_i) - \sum_{i=1}^{n} x_i^2 \tag{E.17}$$

$$\text{subject to} \quad -1 \leq x_i \leq 1, i \in \{1, 2, \ldots, n\}. \tag{E.18}$$

The global maxima are located at the origin with the function values 0.20 and 0.40 for $n = 2$ and $n = 4$, respectively.

10. Dekkers and Aarts Problem (DA)

$$\min_x f(x) = \quad 10^5x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-5}(x_1^2 + x_2^2)^4 \tag{E.19}$$

$$\text{subject to} \qquad -20 \leq x_1, x_2 \leq 20. \tag{E.20}$$

The origin is a local minimizer, but there are two global minimizers located at $x^* = (0, 15)$ and $(0, -15)$ with $f(x^*) = -24776.518$.

11. Easom Problem (EP)

$$\min_x f(x) = \quad -\cos(x_1)\cos(x_2)\exp\left(-(x_1 - \pi)^2 - (x_2 - \pi)^2\right) \tag{E.21}$$

$$\text{subject to} \qquad -10 \leq x_1, x_2 \leq 10. \tag{E.22}$$

The minimum value is located at (π, π) with $f(x^*) = -1$. The function value rapidly approaches zero, when away from (π, π).

12. Epistatic Michalewicz Problem (EM)

$$\min_x f(x) = -\sum_{i=1}^{n} \sin(y_i) \left(\sin\left(\frac{i y_i^2}{\pi} \right) \right)^{2m}, \tag{E.23}$$

$$\text{subject to} \quad 0 \leq x_i \leq \pi, i \in \{1, 2, \ldots, n\}, \tag{E.24}$$

where

$$y_i = \begin{cases} x_i \cos(\theta) - x_{i+1} \sin(\theta), & i = 1, 3, 5, \ldots, < n \\ x_i \sin(\theta) + x_{i+1} \cos(\theta), & i = 2, 4, 6, \ldots, < n \\ x_i, & i = n \end{cases}, \tag{E.25}$$

and $\theta = \frac{\pi}{6}$, $m = 10$.

The number of local minima is not known but the global minimizer is presented in Table E.1.

Tab. E.1: Epistatic Michalewicz's global optimizers

n	$f(x^*)$	x^*
5	-4.687658	(2.693,0.259,2.074,1.023,1.720)
10	-9.660152	(2.693,0.259,2.074,1.023,2.275,0.500,2.138,0.794,2.219,0.533)

13. Exponential Problem (EXP)

$$\max_x f(x) = \exp\left(-0.5 \sum_{i=1}^{n} {x_i}^2 \right) \tag{E.26}$$

$$\text{subject to} \quad -1 \leq x_i \leq 1, i \in \{1, 2, \ldots, n\}. \tag{E.27}$$

The optimal value $f(x^*) = 1$ is located at the origin. Our tests were performed with $n = 10, 20$.

14. Goldstein and Price (GP)

$$\min_x f(x) = [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \tag{E.28}$$
$$\times [30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$$

$$\text{subject to} \quad -2 \leq x_1, x_2 \leq 2. \tag{E.29}$$

There are four local minima and the global minima is located at $x^* = (0, -1)$, with $f(x^*) = 3$.

15. Griewank Problem (GW)

$$\min_x f(x) = \quad 1 + \tfrac{1}{4000}\sum_{i=1}^{n} {x_i}^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{E.30}$$

$$\text{subject to} \quad -600 \leq x_i \leq 600, i \in \{1, 2, \ldots, n\}. \tag{E.31}$$

The function has a global minimum located at $x^* = (0, 0, \ldots, 0)$ with $f(x^*) = 0$. Number of local minima for arbitrary n is unknown, but in the two dimensional case there are some 500 local minima. Tests were performed for $n = 10$.

16. Gulf Research Problem (GRP)

$$\min_x f(x) = \quad \sum_{i=1}^{99} \left[\exp\left(-\frac{(u_i - x_2)^{x_3}}{x_1}\right) - 0.01 \times i\right]^2, \tag{E.32}$$

$$\text{subject to} \quad 0.1 \leq x_1 \leq 100, 0 \leq x_2 \leq 25.6, \text{ and } 0 \leq x_3 \leq 5, \tag{E.33}$$

where $u_i = 25 + [-50 \ln(0.01 \times i)]^{1/1.5}$ This problem has a global minimizer at $(50, 25, 1.5)$ with $f(x^*) = 0$.

17. Hartman 3 Problem (H3)

$$\min_x f(x) = \quad -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right] \tag{E.34}$$

$$\text{subject to} \quad 0 \leq x_j \leq 1, j \in \{1, 2, 3\} \tag{E.35}$$

with constants a_{ij}, p_{ij} and c_i given in Table E.2. There are four local minima, $x_{local} \approx (p_{i1}, p_{i2}, p_{i3})$ with $f(x_{local}) \approx -c_i$. The global minimum is located at

$$x^* \approx (0.114614, 0.555649, 0.852547)$$

with $f(x^*) \approx -3.862782$.

18. Hartman 6 Problem (H6)

$$\min_x f(x) = \quad -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right] \tag{E.36}$$

$$\text{subject to} \quad -0 \leq x_j \leq 1, j \in \{1, \ldots, 6\}, \tag{E.37}$$

with constants a_{ij} and c_i given in Table E.3 and constants p_{ij} in Table E.4. There are four local minima, $x_{local} \approx (p_{i1}, \ldots, p_{i6})$ with $f(x_{local}) \approx -c_i$. The global minimum is located at $x^* \approx (0.201690, 0.150011, 0.476874, 0.275332, 0.311652, 0.657301)$ with $f(x^*) \approx -3.322368$.

Tab. E.2: Data for Hartman 3 problem

i	c_i	a_{ij}			p_{ij}		
		$j=1$	2	3	$j=1$	2	3
1	1	3	10	30	0.3689	0.117	0.2673
2	1.2	0.1	10	35	0.4699	0.4387	0.747
3	3	3	10	30	0.1091	0.8732	0.5547
4	3.2	0.1	10	35	0.03815	0.5743	0.8828

Tab. E.3: Data for Hartman 6 problem

i	c_i	a_{ij}					
		$j=1$	2	3	4	5	6
1	1	10	3	17	3.5	1.7	8
2	1.2	0.05	10	17	0.1	8	14
3	3	3	3.5	1.7	10	17	8
4	3.2	17	8	0.05	10	0.1	14

Tab. E.4: Data for Hartman 6 problem

i	p_{ij}					
	$j=1$	2	3	4	5	6
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	0.2348	0.1451	0.3522	0.2883	0.3047	0.665
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

19. Helical Valley Problem (HV)

$$\min_x f(x) = \quad 100\left[(x_2 - 10\theta)^2 + (\sqrt{(x_1^2 + x_2^2)} - 1)^2\right] + x_3^2 \tag{E.38}$$

$$\text{subject to} \qquad -10 \leq x_1, x_2, x_3 \leq 10 \tag{E.39}$$

where

$$\theta = \begin{cases} \frac{1}{2\pi}\tan^{-1}\frac{x_2}{x_1}, & \text{if } x_1 \geq 0 \\ \frac{1}{2\pi}\tan^{-1}\frac{x_2}{x_1} + \frac{1}{2}, & \text{if } x_1 < 0 \end{cases} \tag{E.40}$$

This is a steep-sided valley which follows a helical path. The minimum is located at $x^* = (1, 0, 0)$ with $f(x^*) = 0$.

20. Hosaki Problem (HSK)

$$\min_x f(x_1, x_2) = \quad \left(1 - 8x_1 + 7x_1^2 - \tfrac{7}{3}x_1^3 + \tfrac{1}{4}x_1^4\right) x_2^2 \exp(-x_2) \tag{E.41}$$

$$\text{subject to} \qquad 0 \leq x_1 \leq 5\,, 0 \leq x_2 \leq 6. \tag{E.42}$$

There are two minima of which the global minimum is $f(x^*) \approx -2.3458$ with $x^* = (4, 2)$.

21. Kowalik Problem (KL)

$$\min_x f(x) = \quad \sum_{i=1}^{11}\left(a_i - \frac{x_1(1 + x_2 b_i)}{(1 + x_3 b_i + x_4 b_i^2}\right)^2 \tag{E.43}$$

$$\text{subject to} \quad 0 \leq x_i \leq 0.42, i \in \{1, 2, 3, 4\}. \tag{E.44}$$

The values for a_i and b_i are given in Table E.5:

Tab. E.5: Data for Kowalik problem

i	1	2	3	4	5	6	7	8	9	10	11
a_i	0.1957	0.1947	0.1735	0.16	0.0844	0.0627	0.0456	0.0342	0.0323	0.0235	0.0246
b_i	0.25	0.50	1.0	2.0	4.0	6.0	8.0	10.0	12.0	14.0	16.0

This is a least squares problem with a global optimal value $f(x^*) \approx 3.0748 \times 10^{-4}$ located at $x^* \approx (0.192, 0.190, 0.123, 0.135)$.

22. Levy and Montalvo 1 Problem (LM1)

$$\min_x f(x) = \frac{\pi}{n}\left(10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1})\right]\right) \quad \text{(E.45)}$$
$$+\frac{\pi}{n}(y_n-1)^2$$

$$\text{subject to} \quad -10 \le x_i \le 10, i \in \{1,2,\ldots,n\} \quad \text{(E.46)}$$

where $y_i = 1 + \frac{1}{4}(x_i+1)$. There are approximately 5^n local minima and the global minimum is known to be $f(x^*) = 0$ with $x^* = (-1,-1,\ldots,-1)$. Our tests were performed with $n = 3$.

23. Levy and Montalvo 2 Problem (LM2)

$$\min_x f(x) = 0.1(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1}] \quad \text{(E.47)}$$
$$+(x_n-1)^2[1+\sin^2(2\pi x_n)])$$

$$\text{subject to} \quad -5 \le x_i \le 5, i \in \{1,2,\ldots,n\}. \quad \text{(E.48)}$$

There are approximately 15^n minima and the global minimizer is known to be $x^* = (1,1,\ldots,1)$ with $f(x^*) = 0$. Our tests were performed with $n = 10$.

24. McCormick Problem (MC)

$$\min_x f(x) = \sin(x_1+x_2) + (x_1-x_2)^2 - (3/2)x_1 + (5/2)x_2 + 1 \quad \text{(E.49)}$$

$$\text{subject to} \quad -1.5 \le x_1 \le 4, -3 \le x_2 \le 3. \quad \text{(E.50)}$$

This problem has a local minimum at $(2.59, 1.59)$ and a global minimum at $(-0.547, -1.547)$ with $f(x^*) \approx -1.9133$.

25. Meyer and Roth Problem (MR)

$$\min_x f(x) = \sum_{i=1}^{5}\left(\frac{x_1 x_3 t_i}{(1+x_1 t_i + x_2 v_i)} - y_i\right)^2 \quad \text{(E.51)}$$

$$\text{subject to} \quad -20 \le x_i \le 20, i \in \{1,2,3\}. \quad \text{(E.52)}$$

This is a least squares problem with minimum value $f(x^*) \approx 0.4 \times 10^{-4}$ located at $x^* \approx (3.13, 15.16, 0.78)$. Table E.6 lists the parameter values of this problem.

Tab. E.6: Data for Meyer and Roth problem

i	t_i	v_i	y_i
1	1.0	1.0	0.126
2	2.0	1.0	0.219
3	1.0	2.0	0.076
4	2.0	2.0	0.126
5	0.1	0.0	0.186

26. Miele and Cantrell Problem (MCP)

$$\min_x f(x) = \left(\exp(x_1) - x_2\right)^4 + 100(x_2 - x_3)^6 + \left(\tan(x_3 - x_4)\right)^4 + {x_1}^8 \quad \text{(E.53)}$$

$$\text{subject to} \qquad -1 \leq x_i \leq 1, i \in \{1, 2, 3, 4\}. \quad \text{(E.54)}$$

The number of local minima is unknown but the global minimizer is located at $x^* = (0, 1, 1, 1)$ with $f(x^*) = 0$.

27. Modified Langerman Problem (ML)

$$\min_x f(x) = -\sum_{j=1}^{5} c_j \cos(\pi d_j) \exp(-d_j/\pi), \quad \text{(E.55)}$$

$$\text{subject to} \qquad 0 \leq x_i \leq 10, i \in \{1, 2, \ldots, n\}, \quad \text{(E.56)}$$

where $d_j = \sum_{i=1}^{n} (x_i - a_{ji})^2$. The test used $n = 10$. The constants c_j and a_{ji} are given in Table E.7.

Tab. E.7: Data for modified Langerman problem

j	c_j	a_{ji}									
		$i = 1$	2	3	4	5	6	7	8	9	10
1	0.806	9.681	0.667	4.783	9.095	3.517	9.325	6.544	0.211	5.122	2.020
2	0.517	9.400	2.041	3.788	7.931	2.882	2.672	3.568	1.284	7.033	7.374
3	0.100	8.025	9.152	5.114	7.621	4.564	4.711	2.996	6.126	0.734	4.982
4	0.908	2.196	0.415	5.649	6.979	9.510	9.166	6.304	6.054	9.377	1.426
5	0.965	8.074	8.777	3.467	1.867	6.708	6.349	4.534	0.276	7.633	1.567

The number of local minima is not known, but the global minima are shown in Table E.8.

Tab. E.8: Global optimizers for modified Langerman problem

n	$f(x^*)$	x^*
5	-0.965	(8.074, 8.777, 3.467, 1.867, 6.708)
10	-0.965	(8.074, 8.777, 3.467, 1.867, 6.708, 6.349, 4.534, 0.276, 7.633, 1.567)

28. Modified Rosenbrock Problem (MRP)

$$\min_x f(x) = \quad 100(x_2 - {x_1}^2)^2 + \left[6.4(x_2 - 0.5)^2 - x_1 - 0.6\right]^2 \tag{E.57}$$

$$\text{subject to} \qquad -5 \leq x_1, x_2 \leq 5. \tag{E.58}$$

This function has two global minima each with $f(x^*) = 0$ (corresponding to the intersection of two parabolas) and a local minimum (where the parabolas approach without intersection). The global minima are located at $x^* \approx (0.3412, 0.1164), (1, 1)$.

29. Multi-Gaussian Problem (MGP)

$$\max_x f(x) = \quad \sum_{i=1}^{5} a_i \exp\left(-((x_1 - b_i)^2 + (x_2 - c_i)^2)/{d_i}^2\right) \tag{E.59}$$

$$\text{subject to} \qquad -2 \leq x_1, x_2 \leq 2. \tag{E.60}$$

The function has one global maximum at $x^* \approx (-0.01356, -0.01356)$ with $f(x^*) \approx 1.29695$. There are also 4 other local maxima and a saddle point. Values for the parameters a_i, b_i, c_i, and d_i are given in Table E.9.

Tab. E.9: Data for multi-Gaussian problem

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

30. Neumaier 2 Problem (NF2)

$$\min_x f(x) = \qquad \sum_{k=1}^{n} \left(b_k - \sum_{i=1}^{n} {x_i}^k \right)^2 \tag{E.61}$$

$$\text{subject to} \quad 0 \leq x_i \leq n, i \in \{1, 2, \ldots, n\}. \tag{E.62}$$

We consider a case when $n = 4$ and $b = (8, 18, 44, 114)$. The global minimum is $f(1, 2, 2, 3) = 0$.

31. Neumaier 3 Problem (NF3)

$$\min_x f(x) = \sum_{i=1}^{n}(x_i - 1)^2 - \sum_{i=2}^{n} x_i x_{i-1} \tag{E.63}$$

$$\text{subject to} \quad -n^2 \leq x_i \leq n^2, i \in \{1, 2, \ldots, n\}. \tag{E.64}$$

The case considered here is $n = 10$. The number of local minima is not known, but the global minima can be expressed as:

$$f(x^*) = -\frac{n(n+4)(n-1)}{6}, \quad x_i^* = i(n+1-i).$$

The global minima for some values of n are presented below.

Tab. E.10: Global minima for Neumaier 3 problem

n	10	15	20	25	30
$f(x^*)$	-210	-665	-1520	-2900	-4930

32. Odd Square Problem (OSP)

$$\min_x f(x) = -(1.0 + 0.2d/(D + 0.01)) \cos(D\pi)\, e^{-D/2\pi} \tag{E.65}$$

$$\text{subject to} \quad -15 \leq x_i \leq 15, i \in \{1, 2, \ldots, 20\} \tag{E.66}$$

where

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

and

$$\underline{b} = (1, 1.3, 0.8, -0.4, -1.3, 1.6, -2, -6, 0.5, 1.4), b_{10+i} = b_i, i = 1, 2, \cdots, 10$$

The number of local minima for a given n is not known but the global minimum is known to be $f(x^*) \approx -1.143833$, $x^* \cong \vec{b}$ (many solutions near $\underline{b}$). We used $n = 10$ in our experiment.

33. Paviani Problem (PP)

$$\min_x f(x) = \sum_{i=1}^{10} \left[(\ln(x_i - 2))^2 + (\ln(10 - x_i))^2 \right] - \left(\prod_{i=1}^{10} x_i \right)^{0.2} \tag{E.67}$$

$$\text{subject to} \quad 2 \leq x_i \leq 10, i \in \{1, 2, \ldots, 10\}. \tag{E.68}$$

This function has a global minimizer at $x_i^* \approx 9.351$ for all i, with $f(x^*) \approx -45.778$.

34. Periodic Problem (PRD)

$$\min_x f(x) = 1 + \sin^2 x_1 + \sin^2 x_2 - 0.1 \exp(-{x_1}^2 - {x_2}^2) \tag{E.69}$$

$$\text{subject to} \quad -10 \leq x_1, x_2 \leq 10. \tag{E.70}$$

There are 49 local minima all with minimum values 1 and global minimum located at $x^* = (0, 0)$ with $f(x^*) = 0.9$.

35. Powell's Quadratic Problem (PWQ)

$$\min_x f(x) = (x_1 + 10x_1)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \tag{E.71}$$

$$\text{subject to} \quad -10 \leq x_i \leq 10, i \in \{1, 2, 3, 4\}. \tag{E.72}$$

This is a unimodal function with $f(x^*) = 0$, $x^* = (0, 0, 0, 0)$. The minimizer is difficult to obtain with accuracy as the Hessian matrix at the optimum is singular.

36. Price's Transistor Modelling Problem (PTM)

$$\min_x f(x) = \gamma^2 + \sum_{k=1}^{4} ({\alpha_k}^2 + {\beta_k}^2) \tag{E.73}$$

$$\text{subject to} \quad -10 \leq x_i \leq 10, i \in \{1, 2, \ldots, 9\}, \tag{E.74}$$

where

$$\begin{aligned}
\alpha_k =& (1 - x_1 x_2) x_3 \{\exp[x_5(g_{1k} - g_{3k} x_7 \times 10^{-3} - g_{5k} x_8 \times 10^{-3})] - 1\} - g_{5k} + g_{4k} x_2, \\
\beta_k =& (1 - x_1 x_2) x_4 \{\exp[x_6(g_{1k} - g_{2k} - g_{3k} x_7 \times 10^{-3} + g_{4k} x_9 \times 10^{-3})] - 1\} \\
& - g_{5k} x_1 + g_{4k}, \\
\gamma =& x_1 x_3 - x_2 x_4.
\end{aligned}$$

The values of g_{ik} are given in Table E.11.

The global minimum occurs very close to $(0.9, 0.45, 1, 2, 8, 8, 5, 1, 2)$ with $f(x^*) = 0$. The number of local minima is unknown.

Tab. E.11: Data for Price's transistor modelling problem

i	g_{ik}			
	$k=1$	2	3	4
1	0.485	0.752	0.869	0.982
2	0.369	1.254	0.703	1.455
3	5.2095	10.0677	22.9274	20.2153
4	23.3037	101.779	111.461	191.267
5	28.5132	111.8467	134.3884	211.4823

37. Rastrigin Problem (RG)

$$\min_x f(x) = \quad 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i)\right] \tag{E.75}$$

$$\text{subject to} \quad -5.12 \leq x_i \leq 5.12, i \in \{1, 2, \ldots, n\}. \tag{E.76}$$

The total number of minima for this function is not exactly known but the global minimizer is located at $x^* = (0, 0, \ldots, 0)$ with $f(x^*) = 0$. For $n = 2$, there are about 50 local minimizers arranged in a lattice like configuration. Our tests were performed with $n = 10$.

38. Rosenbrock Problem (RB)

$$\min_x f(x) = \quad \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right] \tag{E.77}$$

$$\text{subject to} \quad -30 \leq x_i \leq 30, i \in \{1, 2, \ldots, n\}. \tag{E.78}$$

Our tests were performed with $n = 10$. This function is known as the extended Rosenbrock function. It is unimodal, yet due to a saddle point it is very difficult to locate the minimizer $x^* = (1, 1, \ldots, 1)$ with $f(x^*) = 0$.

39. Salomon Problem (SAL)

$$\min_x f(x) = \quad 1 - \cos(2\pi\|x\|) + 0.1\|x\| \tag{E.79}$$

$$\text{subject to} \quad -100 \leq x_i \leq 100 \tag{E.80}$$

where $\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$. The number of local minima (as a function of n) is not known, but the global minimizer is located at $x^* = (0, 0, 0, \ldots, 0)$ with $f(x^*) = 0$. Our tests were performed with $n = 10$.

40. Schaffer 1 Problem (SF1)

$$\min_x f(x) = \quad 0.5 + \frac{\left(\sin\sqrt{x_1^2+x_2^2}\right)^2 - 0.5}{\left(1+0.001\left(x_1^2+x_2^2\right)\right)^2} \tag{E.81}$$

$$\text{subject to} \quad -100 \leq x_1, x_2 \leq 100. \tag{E.82}$$

The number of local minima is not known, but the global minimum is located at $x^* = (0,0)$ with $f(x^*) = 0$.

41. Schaffer 2 Problem (SF2)

$$\min_x f(x) = \quad (x_1^2+x_2^2)^{0.25}\left(\sin^2\left(50(x_1^2+x_2^2)^{0.1}\right)+1\right) \tag{E.83}$$

$$\text{subject to} \quad -100 \leq x_1, x_2 \leq 100. \tag{E.84}$$

The number of local minima is not known, but the global minimum is located at $x^* = (0,0)$ with $f(x^*) = 0$.

42. Schubert Problem (SBT)

$$\min_x f(x) = \quad \prod_{i=1}^n \left(\sum_{j=1}^5 j\cos\left((j+1)x_i + j\right)\right) \tag{E.85}$$

$$\text{subject to} \quad -10 \leq x_i \leq 10, i \in \{1,2,\ldots,n\}. \tag{E.86}$$

Our tests were performed with $n = 2$. The number of local minima for this problem (given n) is not known but for $n = 2$, the function has 760 local minima, 18 of which are global with $f(x^*) \approx -186.7309$. All two dimensional global minimizers are listed in Table E.12:

Tab. E.12: Global optimizers for Schubert problem

x^*				
(-7.0835,4.8580),	(-7.0835,-7.7083),	(-1.4251,-7.0835),	(5.4828,4.8580),	(-1.4251,-0.8003),
(4.8580,5.4828),	(-7.7083,-7.0835),	(-7.0835,-1.4251),	(-7.7083,-0.8003),	(-7.7083,5.4828),
(-0.8003,-7.7083),	(-0.8003,-1.4251),	(-0.8003,4.8580),	(-1.4251,5.4828),	(5.4828,-7.7083),
(4.8580,-7.0835),	(5.4828,-1.4251),	(4.8580,-0.8003)		

43. Schwefel Problem (SWF)

$$\min_x f(x) = \quad -\sum_{i=1}^n x_i \sin\left(\sqrt{|x_i|}\right) \tag{E.87}$$

$$\text{subject to} \quad -500 \leq x_i \leq 500, i \in \{1,2,\ldots,n\}. \tag{E.88}$$

The number of local minima for a given n is not known, but the global minimum value $f(x^*) \approx -418.9829n$ is located at $x^* = (s, s, \ldots, s)$, $s \approx 420.97$. Our tests were performed with $n = 10$.

44. Shekel 5 Problem (S5)

$$\min_x f(x) = \quad -\sum_{i=1}^{5} \frac{1}{\sum_{j=1}^{4} (x_j - a_{ij})^2 + c_i} \tag{E.89}$$

$$\text{subject to} \quad 0 \leq x_j \leq 10, j \in \{1, 2, 3, 4\}, \tag{E.90}$$

with constants a_{ij} and c_j given in Table E.13 below. There are five local minima and the global minimizer is located at $x^* = (4.00, 4.00, 4.00, 4.00)$ with $f(x^*) \approx -10.1532$.

Tab. E.13: Data for Shekel problem family

	i	a_{ij}				c_i
		$j = 1$	2	3	4	
S5	1	4	4	4	4	0.1
	2	1	1	1	1	0.2
	3	8	8	8	8	0.2
	4	6	6	6	6	0.4
	5	3	7	3	7	0.4
S7	6	2	9	2	9	0.6
	7	5	5	3	3	0.3
S10	8	8	1	8	1	0.7
	9	6	2	6	2	0.5
	10	7	3.6	7	3.6	0.5

45. Shekel 7 Problem (S7)

$$\min_x f(x) = \quad -\sum_{j=1}^{7} \frac{1}{\sum_{i=1}^{4} (x_j - a_{ij})^2 + c_i} \tag{E.91}$$

$$\text{subject to} \quad 0 \leq x_j \leq 10, j \in \{1, 2, 3, 4\}, \tag{E.92}$$

with constants a_{ij} and c_j given in Table E.13. There are seven local minima and the global minimizer is located at $x^* = (4.00, 4.00, 4.00, 4.00)$ with $f(x^*) \approx -10.4029$.

46. Shekel 10 Problem (S10)

$$\min_x f(x) = \quad -\sum_{j=1}^{10} \frac{1}{\sum_{i=1}^{4} (x_j - a_{ij})^2 + c_i} \tag{E.93}$$

$$\text{subject to} \quad 0 \le x_j \le 10, j \in \{1, 2, 3, 4\} \tag{E.94}$$

with constants a_{ij} and c_j given in Table E.13. There are 10 local minima and the global minimizer is located at $x^* = (4.00, 4.00, 4.00, 4.00)$ with $f(x^*) \approx -10.5364$.

47. Shekel's Foxholes (FX)

$$\min_x f(x) = \quad -\sum_{j=1}^{30} \frac{1}{c_j + \sum_{i=1}^{n} (x_i - a_{ji})^2} \tag{E.95}$$

$$\text{subject to} \quad 0 \le x_i \le 10, i \in \{1, 2, \ldots, 10\}. \tag{E.96}$$

Our tests were performed with $n = 5$ and 10. The constants c_j and a_{ji} are given in Table E.14. The number of local minima is not known, but the global minima are presented in Table E.15.

48. Sinusoidal Problem (SIN)

$$\min_x f(x) = \quad -\left[A \prod_{i=1}^{n} \sin(x_i - z) + \prod_{i=1}^{n} \sin(B(x_i - z))\right] \tag{E.97}$$

$$\text{subject to} \quad 0 \le x_i \le 180, i \in \{1, 2, \ldots, n\}. \tag{E.98}$$

The variable x is in degrees. Parameter A affects the amplitude of the global optimum; B affects the periodicity and hence the number of local minima; z shifts the location of the global minimum; and n indicates the dimension. Our tests were performed with $A = 2.5, B = 5, z = 30$, and $n = 10$ and 20. The location of the global solution is at $x^* = (90 + z, 90 + z, \ldots, 90 + z)$ with the global optimum value of $f(x^*) = -(A + 1)$. The number of local minima increases dramatically in dimension, and when $B = 5$ the number of local minima is equal to:

$$\sum_{i=0}^{\lfloor n/2 \rfloor} \left(\frac{n!}{(n-2i)!(2i)!} 3^{n-2i} 2^{2i} \right). \tag{E.99}$$

Tab. E.14: Data for Shekel's foxholes problem

j	c_j	a_{ji}									
		$i=1$	2	3	4	5	6	7	8	9	10
1	0.806	9.681	0.667	4.783	9.095	3.517	9.325	6.544	0.211	5.122	2.020
2	0.517	9.400	2.041	3.788	7.931	2.882	2.672	3.568	1.284	7.033	7.374
3	0.100	8.025	9.152	5.114	7.621	4.564	4.711	2.996	6.126	0.734	4.982
4	0.908	2.196	0.415	5.649	6.979	9.510	9.166	6.304	6.054	9.377	1.426
5	0.965	8.074	8.777	3.467	1.863	6.708	6.349	4.534	0.276	7.633	1.567
6	0.669	7.650	5.658	0.720	2.764	3.278	5.283	7.474	6.274	1.409	8.208
7	0.524	1.256	3.605	8.623	6.905	4.584	8.133	6.071	6.888	4.187	5.448
8	0.902	8.314	2.261	4.224	1.781	4.124	0.932	8.129	8.658	1.208	5.762
9	0.531	0.226	8.858	1.420	0.945	1.622	4.698	6.228	9.096	0.972	7.637
10	0.876	7.305	2.228	1.242	5.928	9.133	1.826	4.060	5.204	8.713	8.247
11	0.462	0.652	7.027	0.508	4.876	8.807	4.632	5.808	6.937	3.291	7.016
12	0.491	2.699	3.516	5.874	4.119	4.461	7.496	8.817	0.690	6.593	9.789
13	0.463	8.327	3.897	2.017	9.570	9.825	1.150	1.395	3.885	6.354	0.109
14	0.714	2.132	7.006	7.136	2.641	1.882	5.943	7.273	7.691	2.880	0.564
15	0.352	4.707	5.579	4.080	0.581	9.698	8.542	8.077	8.515	9.231	4.670
16	0.869	8.304	7.559	8.567	0.322	7.128	8.392	1.472	8.524	2.277	7.826
17	0.813	8.632	4.409	4.832	5.768	7.050	6.715	1.711	4.323	4.405	4.591
18	0.811	4.887	9.112	0.170	8.967	9.693	9.867	7.508	7.770	8.382	6.740
19	0.828	2.440	6.686	4.299	1.007	7.008	1.427	9.398	8.480	9.950	1.675
20	0.964	6.306	8.583	6.084	1.138	4.350	3.134	7.853	6.061	7.457	2.258
21	0.789	0.652	2.343	1.370	0.821	1.310	1.063	0.689	8.819	8.833	9.070
22	0.360	5.558	1.272	5.756	9.857	2.279	2.764	1.284	1.677	1.244	1.234
23	0.369	3.352	7.549	9.817	9.437	8.687	4.167	2.570	6.540	0.228	0.027
24	0.992	8.798	0.880	2.370	0.168	1.701	3.680	1.231	2.390	2.499	0.064
25	0.332	1.460	8.057	1.336	7.217	7.914	3.615	9.981	9.198	5.292	1.224
26	0.817	0.432	8.645	8.774	0.249	8.081	7.461	4.416	0.652	4.002	4.644
27	0.632	0.679	2.800	5.523	3.049	2.968	7.225	6.730	4.199	9.614	9.229
28	0.883	4.263	1.074	7.286	5.599	8.291	5.200	9.214	8.272	4.398	4.506
29	0.608	9.496	4.830	3.150	8.270	5.079	1.231	5.731	9.494	1.883	9.732
30	0.326	4.138	2.562	2.532	9.661	5.611	5.500	6.886	2.341	9.699	6.500

Tab. E.15: Global optimizers for Shekel's foxholes problem

n	$f(x^*)$	x^*
5	-10.4056	(8.025, 9.152, 5.114, 7.621, 4.564)
10	-10.2088	(8.025, 9.152, 5.114, 7.621, 4.564, 4.771, 2.996, 6.126, 0.734, 4.982)

49. Storn's Tchebychev Problem (ST)

$$\min_x f(x) = p_1 + p_2 + p_3, \tag{E.100}$$

where

$$p_1 = \begin{cases} (u-d)^2 & \text{if } u < d \\ 0 & \text{if } u \geq d \end{cases} \qquad u = \sum_{i=1}^{n} (1.2)^{n-i} x_i$$

$$p_2 = \begin{cases} (v-d)^2 & \text{if } v < d \\ 0 & \text{if } v \geq d \end{cases} \qquad v = \sum_{i=1}^{n} (-1.2)^{n-i} x_i$$

$$p_3 = \sum_{j=0}^{m} \begin{cases} (w_j - 1)^2 & \text{if } w_j > 1 \\ (w_j + 1)^2 & \text{if } w_j < -1 \\ 0 & \text{if } -1 \leq w_j \leq 1 \end{cases} \qquad w_j = \sum_{i=1}^{n} \left(\frac{2j}{m} - 1 \right)^{n-i} x_i,$$

for $n = 9$: $x_i \in [-128, 128]^n$, $d = 72.661$, and $m = 60$

for $n = 17$: $x_i \in [-32768, 32768]^n$, $d = 10558.145$, and $m = 100$.

The number of local minima is not known but the global minimum is known to be as shown in Table E.16. Our tests were performed with $n = 9$.

Tab. E.16: Global optimizers for Storn's Tchebychev problem

n	$f(x^*)$	x^*
9	0	(128, 0, -256, 0, 160, 0, -32, 0, 1)
17	0	(32768, 0, -1331072, 0, 21299, 0, -180224, 84480, 0, -2154, 0, 2688, 0, -128, 0, 1)

50. Wood's Problem (WP)

$$\min_x f(x) = \begin{aligned} & 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 \\ & +10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1) \end{aligned} \tag{E.101}$$

$$\text{subject to} \qquad -10 \leq x_i \leq 10, i \in \{1, 2, 3, 4\}. \tag{E.102}$$

The function has a saddle near $(1, 1, 1, 1)$. The only minimum is located at $x^* = (1, 1, 1, 1)$ with $f(x^*) = 0$.

51. Zakharov (ZAK)

$$\min_x f(x) = \left(\sum_{j=1}^{n} x_j^2\right) + \left(\sum_{j=1}^{n} 0.5jx_j\right)^2 + \left(\sum_{j=1}^{n} 0.5jx_j\right)^4 \quad \text{(E.103)}$$

$$\text{subject to} \quad -5 < x_j < 10, j = 1, \cdots, n. \quad \text{(E.104)}$$

The function has several local minima (exact number unspecified in the literature). The global minimum is located at $x^* = (0, \cdots, 0)$ with $f(x^*) = 0$. Our tests were performed with $n = 10$.

52. Problem SBT3

$$\min_x f(x) = \frac{pi}{n}\left\{10\sin^2(\pi x_1) + \sum_{i=1}^{n-1}\left[(x_i - 1)^2(1 + 10\sin^2(\pi x_{i+1}))\right]\right\} \quad \text{(E.105)}$$
$$+\frac{\pi}{n}(x_i - 1)^2$$

$$\text{subject to} \quad -10 \leq x_i \leq 10, i \in \{1, 2, \ldots, n\}, \quad \text{(E.106)}$$

This function has roughly 10^n local minimizers and a unique global minimizer $f^* = 0$ at $x^* = (1, \cdots, 1)$. Our experiments were performed with $n = 3$ and 5.