LAPPEENRANTA
UNIVERSITY OF TECHNOLOGY

Jani Rönkkönen

CONTINUOUS MULTIMODAL GLOBAL OPTIMIZATION WITH DIFFERENTIAL EVOLUTION-BASED METHODS

Thesis for the degree of Doctor of Science (Technology) to be presented with due permission for public examination and criticism in the Auditorium of the Student Union House at Lappeenranta University of Technology, Lappeenranta, Finland on the 8th of December, 2009, at noon.

Acta Universitatis
Lappeenrantaensis
363

Supervisor	Professor Ville Kyrki Machine Vision and Pattern Recognition Laboratory Department of Information Technology Faculty of Technology Management Lappeenranta University of Technology Finland
Reviewers	Professor Kalyanmoy Deb Kanpur Genetic Algorithms Laboratory Department of Mechanical Engineering Indian Institute of Technology Kanpur India
	Professor Daniela Zaharie Faculty of Mathematics and Computer Science West University of Timisoara Romania
Opponent	Professor Tommi Kärkkäinen Department of Mathematical Information Technology University of Jyväskylä Finland

ISBN 978-952-214-851-3
ISBN 978-952-214-852-0 (PDF)
ISSN 1456-4491

Lappeenrannan teknillinen yliopisto
Digipaino 2009

To my loving wife Laura and our two magnificent boys, Miska and Otso.

Preface

The work presented in this thesis has been done in 2003-2009 at department of information technology, Lappeenranta University of Technology. Most of the time I worked as an assistant, combining research activities with teaching responsibilities, which have provided challenging and varying job assignments. These years have been very rich in my life in both professionally and in private life and I would like to express my gratitude to all who have directly or indirectly supported my project with the thesis.

First I would like to thank my supervisor Ville Kyrki, whose excellent guidance and support have been invaluable during my work with the thesis. Despite his tight schedule, Ville always manages to find time for his students, which is greatly appreciated. I would also like to thank Jouni Lampinen, who introduced me to the world of evolutionary computation and scientific research. Jouni also worked as my supervisor in the early stages of my Ph.D. studies and has co-authored most of my publications. These have been invaluable contributions to my thesis project and are greatly appreciated.

I would like to also express my gratitude to the other co-authors I have had the privilege of working with: Kenneth Price, Xiaodong Li and Saku Kukkonen. Timo Mantere and Junhong Liu have also worked in the same research group and provided important insights to the evolutionary computation during the early stages of my studies. Jouni Sampo has been of great help with some mathematical problems. The careful and constructive comments from the reviewers, Kalyanmoy Deb and Daniela Zaharie are also gratefully appreciated.

For the financial support I would like to thank the department of information Technology in Lappeenranta University of Technology and Heikki Kälviäinen as well as the Research Foundation of Lappeenranta University of Technology (Lappeenrannan teknillisen yliopiston tukisäätiö).

For maintaining the motivation inspiring working environment is essential. Such an environment was provided by the staff of the department information technology and it has been a pleasure working there. During these years, I have gained many friends among the great people of the department. Especially I will be missing the interesting and diverse discussions during the coffee and lunch breaks with Ville, Leena, Jarmo, Jussi, Jarkko, Olli, Toni, Tuomas, Teemu&Teemu, Jukka, Janne, Pekka, Saku, Tomi, Ossi, Timo, Kari, Jouni, Ilmari and all the rest of the group. Also a lot of useful information has been passed during these breaks.

Board gaming and game design have also been a great ways of relieving the occasional stress caused by the thesis project. I would like to thank the members of gaming club Louhi and all the other gamers and game testers who have provided me the chance to pursue these interests regularly during these years.

Finally I would like to express my gratitude to the people closest to me. I have always been free to pursue my own path, for which I would like to thank my mother Kerttu. Thank you also for my sister Ella for all the interesting ideas and discussions and my parents-in-law for your support to the thesis project through offering your help in child-care. Last but not the least I would like to thank my family, Laura, Miska and Otso, for your support and love and simply being there.

“In the struggle for survival, the fittest win out
at the expense of their rivals because they succeed
in adapting themselves best to their environment.”,
– Charles Darwin

“Anyone who attempts to generate
random numbers by deterministic means is,
of course, living in a state of sin.”,
– John von Neumann

Lappeenranta, November 2009

Jani Rönkkönen

Abstract

Jani Rönkkönen
Continuous Multimodal Global Optimization with Differential Evolution-Based Methods
Lappeenranta, 2009
168 p.

Acta Universitatis Lappeenrantaensis 363
Diss. Lappeenranta University of Technology

ISBN 978-952-214-851-3
ISBN 978-952-214-852-0 (PDF)
ISSN 1456-4491

Metaheuristic methods have become increasingly popular approaches in solving global optimization problems. From a practical viewpoint, it is often desirable to perform multimodal optimization which, enables the search of more than one optimal solution to the task at hand. Population-based metaheuristic methods offer a natural basis for multimodal optimization. The topic has received increasing interest especially in the evolutionary computation community. Several niching approaches have been suggested to allow multimodal optimization using evolutionary algorithms.

Most global optimization approaches, including metaheuristics, contain global and local search phases. The requirement to locate several optima sets additional requirements for the design of algorithms to be effective in both respects in the context of multimodal optimization. In this thesis, several different multimodal optimization algorithms are studied in regard to how their implementation in the global and local search phases affect their performance in different problems. The study concentrates especially on variations of the Differential Evolution algorithm and their capabilities in multimodal optimization. To separate the global and local search search phases, three multimodal optimization algorithms are proposed, two of which hybridize the Differential Evolution with a local search method.

As the theoretical background behind the operation of metaheuristics is not generally thoroughly understood, the research relies heavily on experimental studies in finding out the properties of different approaches. To achieve reliable experimental information, the experimental environment must be carefully chosen to contain appropriate and adequately varying problems. The available selection of multimodal test problems is, however, rather limited, and no general framework exists. As a part of this thesis, such a framework for generating tunable test functions for evaluating different methods of multimodal optimization experimentally is provided and used for testing the algorithms. The results demonstrate that an efficient local phase is essential for creating efficient multimodal optimization algorithms. Adding a suitable global phase has the potential to boost the performance significantly, but the weak local phase may invalidate the advantages gained from the global phase.

Keywords: global optimization, metaheuristics, evolutionary computation, evolutionary algorithms, Differential Evolution, mutation, multimodal optimization, niching, crowding, test function generator, hybrid methods, memetic algorithms, local optimization, gradient descent

UDC 519.863 : 004.021

Symbols and abbreviations

$\otimes$	Component-wise multiplication of vectors
α^*	Optimal value for α, where α can be a parameter or a performance measure
α	Amplitude of the sampling function in the cosine family
α_{scale}	Scaling factor for sharing function
β	Shape of the optimum slope (hump family)
$\vec{\gamma}$	Generic vector
δ	Initial bracketing length for GD
η	Scaling used in estimating the gradient with GD
ε	Precision used to define optima
κ	Capacity of each niche with clearing
λ	Number of offspring generated
μ	Number of parents
ν	Degree of the Bezier curve, defines the number of the control points for $\vec{P}$
σ	Parameter controlling the standard deviation in dither, jitter and ajitter mutation operators
σ_{rad}	Radius parameter in different niching methods
σ_ζ,σ_η	Standard deviations for ω_ζ and ω_η
ω_ζ,ω_η	Zero-mean normally distributed variables used by the G3-PCX algorithm
$\mathbf{A}$	Generic matrix
$\mathbf{B}$	Rotated symmetric matrix corresponding to $\mathbf{C}$
$\vec{b}$	Rotated point corresponding to $\vec{x}$
c	Number of niches or subpopulations maintained by a niching method
$\mathbf{C}$	Symmetric matrix (quadratic family)
c^{var}	Expected effect of the variation operators of DE to the variance of the population
CF	Crowding factor
CR	Crossover rate in DE
D	Dimensionality of a problem
$\bar{D}$	Average of perpendicular distances used by the G3-PCX algorithm
d	Distance measure
dnc	Dynamic niche count
$\vec{e}$	Orthonormal bases used by G3-PCX algorithm
F	Mutation scale factor in the DE
$\vec{F}^{ajit}$	Mutation scaling vector for DE using additive jitter
F^{dith}	Mutation scaling factor for DE using dither
$\vec{F}^{jitt}$	Mutation scaling vector for DE using jitter
f'	First order derivative
f''	Second order derivative
f^c	Clustered shared fitness
f^d	Dynamic shared fitness
f^s	Shared fitness

g	Generation
$\vec{G}$	Vector of positive integers defining the number of global minima in the cosine family
i, j	Generic enumeration variables
l	Number of constraints an optimum sits on
$\vec{L}$	Vector of positive integers defining the number of local minima in the cosine family
m	Minimum number of members in each species of SDE
n	Generic variable defining a count or a number
$N(e, \sigma)$	Random number drawn from normal distribution with expectation e and standard deviation σ
NFE_{max}	Maximum allowed number of function evaluations
NFE_{slope}	Slope of the curve representing the convergence speed in the function of population size
nnp	Population size inside a single niche
NP	Population size
$\mathbf{O}$	Randomly generated angle preserving orthogonal linear transformation matrix
$\vec{P}$	Control point vector for a Bezier curve
PX	Probability to select the first operator in either/or algorithm
$\vec{q}$	Vector for defining an optimum location (quadratic family)
r	Index for a randomly selected population member in DE
$\mathbb{R}$	Real numbers
r^{rad}	Basin radius of an optimum (hump family)
$rand[0, 1]$	Uniformly distributed random number in the range [0,1]
sh	Sharing function
$\vec{u}$	Trial solution vector
$\vec{v}$	Mutated vector in DE
$\vec{x}$	Population member vector
$\vec{x}^L$	Vector of lower box constraints
$\vec{x}^U$	Vector of upper box constraints
$\vec{y}$	Stretched point corresponding to $\vec{b}$
v	$f(\vec{x}^*)$ for one optimum

AJIT	DELL using ajitter global mutation
AJIG	DELG using ajitter global mutation
AOA	Area of attraction
CG	Conjugate gradient
CMA-ES	Covariance matrix adaptation evolution strategy
CPT	Complexity per trial
CRDE	Crowding DE
DE	Differential Evolution
DEGS	DE using global selection
DECG	CRDE hybridized with gradient descent
DELG	DELS hybridized with gradient descent
DELL	DELS with local mutation

DELS	DE using local selection
DITG	DELG using dither global mutation
DITH	DELL using dither global mutation
EA	Evolutionary algorithm
EP	Evolutionary programming
ES	Evolution strategy
FPS	Fitness proportional selection
G3	Generalized generation gap
GA	Genetic algorithm
GD	Gradient Descent
GO	Global optimization
GRGD	Grid-based gradient descent
HCRDE	CRDE using hill-valley detection
HDECG	DECG using hill-valley detection
HM	Harmony Memory
JITG	DELG using jitter global mutation
JITT	DELL using jitter global mutation
LOVR	Local optima value range
LS	Local search
MCMC	Markov chain Monte Carlo
MCDE	Multipopulation crowding DE
MMDE	Multiresolution multipopulation DE
MSG	Max Set of Gaussians
NFE	Number of function evaluations
NFL	No free lunch (theorems)
NGO	Number of global optima
NLO	Number of local optima
PCX	Parent-centric recombination
PN	Parallel niching
PR	Peak ratio
PSO	Particle swarm optimization
RE	Replacement error
RS	Ranking selection
RSGD	Random start gradient descent
SA	Simulated annealing
SDE	Speciation-based DE
SHDE	Sharing DE
SLS	Stochastic local search
SN	Sequential niching
SP	Success performance
SR	Success rate
std	Standard deviation
TS	Tabu search
URE	Unwanted replacement error
XLS	Crossover-based local search

Contents

Chapter I

Introduction

Global optimization is an important field of study, as the methods can be applied to a wide variety of applications from industrial design to creating art. Often only limited information is available for the problem to be optimized. Additionally, real-world problems are often high-dimensional, contain a large number of optima, and may contain discontinuities. These features pose a severe challenge for global optimization methods.

As a response for this challenge, a class of global optimization algorithms called metaheuristics have emerged. They aim to solve problems by using heuristic procedures which capture and exploit the essential information of the problem without the need for making a priori assumptions of the problem. Metaheuristic methods have become a popular choice as global optimization algorithms, as they have demonstrated an ability to solve difficult real-world problems in a wide variety of applications (see for example [17, 125, 12]).

Evolutionary algorithms (EA) are one of the most popular classes of metaheuristics. Their foundation is in mimicking the evolution process of a population of solutions, which evolves through genetic variation and natural selection. Basic EAs are typically designed for locating a single global optimum. Real-world optimization problems, however, often contain multiple optima. From a practical viewpoint it is often desirable to be able to locate more than one good solution to offer the decision maker options to choose from. In this thesis, the term multimodal optimization refers to the attempt to locate all global optima and optionally also good local optima of a multimodal function. The use of the population makes EAs an interesting candidate for multimodal optimization. To employ this potential, techniques called niching methods have been developed. The general idea of niching is to somehow prevent the whole population from converging to a single area of the search space and thus allow multimodal optimization.

Most global optimization approaches, including metaheuristics, contain global and local search phases. These are especially visible in hybrid methods, which aim to combine different optimization approaches in a way that allows the hybrid to employ the strengths of all parent algorithms, and minimizes their weaknesses. In principle, any combination of algorithms can be hybridized, but usually the best results are achieved by combining algorithms which can offer different strengths to the hybrid. Especially interesting are the combinations of local and global optimizers. The term memetic algorithms refers to methods that combine EAs with local search methods.

As the theoretical background behind the operation of metaheuristics is not generally thoroughly understood, the research relies heavily on experimental studies in learning the properties of different approaches. To achieve reliable experimental information, testing environment must be carefully chosen to contain appropriate and adequately varying problems. Various test function setups and function generators have been presented to allow quality comparisons to be performed. At the moment, most of these are designed for measuring the capability to locate a single optimum, which leaves the present selection of test problems for multimodal optimization approaches quite limited. This hinders experimental studies with such approaches.

1.1 Objectives

This thesis concentrates on studying the features essential for successful multimodal optimization algorithms in the context of continuous optimization, where the problems are represented as a function whose parameter space is real. The main objective is to study the different implementations of global and local search phases, and to clarify their effect on the performance of the algorithms. Additionally, relevant features of the used test problems and their effects on the performance of different approaches are studied. The objective is to identify the reasons behind good or poor performance, and to use this information to develop multimodal optimization algorithms further.

Among the EAs, the Differential Evolution (DE) algorithm has been selected and studied in detail. DE has demonstrated a good performance in a variety of applications [125, 23], it has been designed for continuous optimization, and it is simple to implement. The use of a single base algorithm allows more detailed analysis of the algorithm features, as well as reliable comparison of the niching approaches. Especially the suitability of global and local selection-based DE approaches for multimodal optimization are studied. One of the objectives is to identify the strengths and weaknesses of local selection as a niching method, compared to other selection schemes. Additionally, the possible advantages of using randomization in the mutation operation of the DE algorithm are studied.

To be able to evaluate the multimodal optimization algorithms experimentally, an extended testing environment is required. One of the objectives is also to provide such an environment for the public use, in addition to using it to provide the testing environment for this study. The idea is that by providing a standard environment, comparable studies can be performed by other researchers, allowing a wider comparison of different multimodal approaches.

1.2 Contributions and publications

The main contribution of this thesis is the increased understanding of the roles of global and local search phases in the performance of multimodal optimization algorithms: An algorithm able to do both effectively, will potentially gain advantage over algorithms which excel only in one or the other. As a result, hybridization offers a promising course for increasing the performance of multimodal optimization algorithms. While hybridization has been studied in the context of locating a single optimum, studies concentrating on multimodal optimization are remarkably rare. Additionally, new understanding is gained about the properties of the DE algorithm in the context of multimodal optimization and different niching methods. Especially the ability of DE to exploit regularity and the effects to performance are revealed. The findings have been partially published in [136, 138].

On a more practical level, three novel DE-based multimodal optimization approaches based on the idea of separating the global and local search phases are proposed. A comparative study performed in [133] demonstrates the potential of DE in solving multimodal problems. However, the parameter study performed in [124] reveals a dependence between the population size (NP) and mutation step length, as well as a drift-bias towards the better population members for the traditional global selection-based DE. The paper also demonstrates that no such biases exist for a version of DE based on a local selection. These findings, along with the results of a study of using randomization in the mutation of DE [134] have been used as a basis for designing the local selection-based DE algorithm capable of multimodal optimization in [135]. The algorithm has been developed further, and two hybrid approaches for multimodal optimization are proposed in [138].

Another important contribution is the framework for generating multimodal test functions, which has been published in [137, 138]. As the selection of test functions for evaluating multimodal optimization algorithms experimentally has been very limited, the introduction of the generator framework allows for more versatile experimental studies to be performed.

In the above mentioned publications, the author has made the majority of the development, writing and and experimental work in all but [134, 124], in which the author performed the implementation and experimental study and had a minor contribution in writing.

1.3 Outline of the thesis

The thesis consists of seven chapters. Chapter 2 provides an overview of continuous global optimization. The chapter is divided into five parts, reflecting the chosen classification of the optimization methods. Chapter 3 offers an overview of the topic of multimodal global optimization and of the related concept of niching. A rarely used way of measuring the computational complexity of the niching methods is used, and the rational behind that is explained. A classification into sequential and parallel niching methods is used. The chapter presents the most important niching methods belonging to both categories, especially concentrating on the parallel niching methods.

The topic of Chapter 4 is experimental evaluation of continuous global optimization algorithms. First, some important exploitable problem features are described, followed by the definition of relevant performance measures. Then an experimental study between two optimization algorithms, Differential Evolution and Generalized Generation Gap is reported. The study fulfills two purposes: It demonstrates the potential of DE in solving multimodal problems, and demonstrates the difficulties related to comparing optimization algorithms through experimental results. Finally, the chapter describes a proposed framework for generating multimodal test functions.

Chapter 5 concentrates on the DE algorithm, describing the basic algorithm and a brief history. A discussion about using a randomized scale factor on the mutation operation follows. Then the focus moves to considering the selection pressure of the algorithm, and the concept of local selection is introduced. The differences between the local and traditional global selection concepts are highlighted and the local selection is generalized to multimodal optimization. Additionally, the chapter presents an overview of niching techniques used with the DE algorithm. Finally, a brief overview of hybridizing the DE algorithm is presented, and two novel hybrid algorithms for multimodal optimization are proposed.

Chapter 6 presents the main experimental setup and results in two parts: In the first part eight multimodal optimization approaches, including the proposed approaches, are compared by using a set of functions generated by the test function framework presented in Chapter 4. The first part concentrates on determining the ability of each approach to locate multiple global optima in different functions and to explain the reasons behind good or poor performance. In the second part, the ability of the most interesting methods to locate and maintain also local optima is studied. Chapter 7 concludes the thesis.

Chapter II

Continuous global optimization

Optimization algorithms (optimizers) can be classified to methods that solely perform local search (LS) and methods designed for global optimization (GO). Global optimization is the practice of searching for the *global optimum* point $\vec{x}^* = \{x_1, \ldots, x_D\}$ which produces an optimal value for a *fitness function* $f(\vec{x})$ subject to box constraints $x_i^L \leq x_i \leq x_i^U$. While the optimization task may be either minimization or maximization of the fitness function, in this thesis all experimental work assumes minimization. Additionally, only functions having a real parameter space ($\vec{x} \in \mathbb{R}^D$) are considered in the context of this thesis. The term *fitness value* or simply fitness is used to refer to the value of $f(\vec{x})$ for a particular solution candidate $\vec{x}$. Local searchers aim to efficiently locate a *local optimum* residing near the starting point. A local optimum is a point which can not be improved by a slight move in the neighborhood but is not necessarily a global optimum. More formally, a local optimum is a point for which the first order derivative $f'(\vec{x}) = 0$ and the second order derivative $f''(\vec{x}) > 0$ (minimum) or $f''(\vec{x}) < 0$ (maximum). In contrast, global optimizers aim to explore the search space extensively enough to locate the actual global optimum. While for some functions, the optima are not well defined due to points of discontinuity, this work focuses on numerical optimization where such limitations are not an issue, as the optimum is then simply the nearest point from the point of discontinuity within the desired numerical precision.

In practice, most, but not all, global optimizers can be considered to consist of a *local* and a *global search phase* [141]. The global phase is responsible for the exploration of the search space and aims to identify the potentially promising areas where the global optimum could be located. For this, information from previous iterations is used through the population or some other memory structure. The local phase aims to increase the efficiency of locating the actual optima on the promising areas. In many algorithms these phases are blended together and are not easily distinguishable.

2.1 Pure global optimization methods

Pure global optimization methods rely on the global search phase and do not contain any local improvement phase. The *exhaustive search* algorithm is the most fundamental of the GO approaches. The idea is simply to go through all points of the search space in a predetermined order. While theoretically the number of points for real coded problems is infinite, in practice it is always possible to implement such an approach by defining the limits in numerical precision. Exhaustive search, as all pure GO methods, is *theoretically convergent*, i.e. it guarantees convergence to a global optimum in finite time. While finite, the time to find the global optimum is still typically too long for practical use. When all possible problems are considered in the sense of the no free lunch (NFL) theorems [179, 180], exhaustive search is an optimal GO approach, because it never searches the same point more than once. This is the only possible assumption which can be made to improve the performance of an optimization algorithm over all possible problems. In this sense the order in which the points are searched is irrelevant.

Typically, however, an optimization algorithm is designed for a small subset of all possible problems. For example, the real-life optimization problems of interest have typically some meaningful structure, so that neighboring points are somehow related and some assumptions can be made from the value of the function in most points on the basis of nearby points. By selecting a subset of problems, the performance of exhaustive search can be improved through modifying the sampling order in which the points are searched. *Grid search* aims to maximize the coverage of each part of the search phase by generating the points in a predetermined grid. First a coarse search is performed, and at each generation the grid gets denser.

Another option to cover the search space is to use *pure random search*, which simply generates random points uniformly from the search space. Compared to exhaustive search, pure random search is not optimal in the sense of NFL theorems, because the same points can be generated multiple times. While the method is also convergent like exhaustive search, the ending condition can not be similarly decided, because the information about when all points have been considered does not exist. The advantage of generating points randomly instead of using a grid or some other predetermined order is, that it offers an unbiased (assuming uniform distribution is used) sample distribution over the search space regardless of the ending condition. If exhaustive search is ended before sampling through all points, the searched points are determined by the used sampling strategy, which is always biased.

Branch and bound methods [160] improve the performance of exhaustive search by analyzing and reducing the search phase during the optimization process. To achieve the reduction, they divide the search space to sub regions and decide the lower bound for any optima inside. Now any region having worse lower bound than the current optimum can be discarded. The performance depends heavily on the accuracy of the lower bound-approximating function. To generate the approximating function, some characteristics of the fitness function must be known. This limits the usability of the branch and bound methods to problems for which the generation of lower bound approximating function is possible. Without the function, branch and bound reduces to exhaustive search.

2.2 Methods for local search

The local search methods aim at locating a local optimum from a given starting point, and they do not contain the global search phase. The idea is to generate a sequence of points which leads to the local optimum as fast as possible. Basically LS algorithms need to devise a strategy for deciding the length of each step and its direction, which decide the efficiency of the algorithm. This section introduces some of the most important LS methods briefly.

The LS methods can be classified to derivative-based methods and direct search methods, according to the method used to decide the search direction. As the name implies, derivative-based methods require derivatives and are very efficient in differentiable functions. Direct search methods do not require derivatives, and sample points from the local neighborhood instead deciding the search direction based on the fitness function values on these points. While the direct search methods are applicable to broader sets of problems, they converge typically significantly slower compared to derivative-based methods in problems where both are applicable. As the derivatives can often be approximated numerically, the set of meaningful problems for which the derivative-based methods are completely inapplicable is small, and thus the direct search LS methods are often the less attractive choice.

2.2.1 Derivative-based methods

Derivative-based methods can be classified further to methods using gradients (first order derivatives) and methods using higher order derivatives. The advantage of using second order derivatives is that step length can be estimated in addition to the direction, in contrast to the gradient which contains only directional information. However, the applicability of derivative-based methods can be extended to a broader set of problems by approximating the gradient numerically. The precision of the approximation then decides the success of the algorithm. Typically the gradient can be estimated accurately enough for continuous functions, but the second order derivative is already much more sensitive and difficult to approximate. This limits the usability of methods based on higher order derivatives severely in functions for which exact derivatives are not available.

Gradient descent (GD), also called steepest descent, algorithms simply descend downhill in the direction of the steepest slope, i.e. the negative gradient direction. Using the gradient direction directly tends to oscillate when the optima shape is a narrow valley [146], which decreases the efficiency of the algorithm. *Conjugate gradient* (CG) methods [146] [121, p. 420] construct the direction as the conjugate to the previous gradient which is closest to the current gradient direction. This decreases the oscillation, and CG is typically more efficient than GD. The (usually minor) downsize of using CG is the requirement to store the previous search direction.

GD and CG refer to the method of deciding the direction, and they leave open the decision of the jump length, which is typically decided by a *line search* procedure. Once the search direction is decided by GD or CG, the line search finds the optimal solution along that search line. The line search procedure typically uses first a *bracketing* method for bounding the area where an optimum is located, and then a fine-tuning method for locating the actual optimum inside the bounded area.

In this thesis, the bracketing routine described in [121, p. 400] is adopted. The idea is simply to locate three points along the search line where the middle one has better fitness value compared to the other two. This means that an optimum is located somewhere between the two outermost points. The bracketing uses the initial point and another point generated along the search line to a distance of the initial bracketing length. The third point is generated in the direction of improving fitness by using the distance of the two points multiplied by the golden ratio $(1 + \sqrt{5})/2$. The process is repeated as long as the new point increases the fitness value by using the differential of the two latest points and replacing the oldest point (which has the worst fitness value) with the generated point. In this thesis, the value of 10^{-3} is always used for the initial bracketing length δ. Because the step length increases with each step, the selected value has only a limited effect on the efficiency and does not decide the success of the bracketing, as long as an appropriately small value is used.

Brent's method [121, p. 402] is an efficient method for locating the actual optima inside the bracketed area of the search line. The method starts from the three points located by bracketing and alternates *inverse parabolic interpolation* and *golden section search* until the optimum is located with a required precision. The golden section search is a reversed version of the bracketing procedure: the longer of the differentials between the middle point and either of the border points is divided by the golden ratio and the points are updated to form a new set of three bracketing points, which now bound the optimum tighter. By repeating this procedure, the actual optimum is eventually found with required precision. The inverse parabolic interpolation fits a parabola through the three points, and the location of the minimum of the parabola is used to approximate the location of the optimum. As parabolic interpolation is very efficient in functions whose area near the optimum is parabolic, the parabolic fit is attempted first. If it fails to produce an acceptable point, Brent's method uses the golden section search instead. The described combination of bracketing and Brent's method is used as a line search operator for all related experimental results in this thesis.

Newton's (or Newton-Rhapson) method [26, p. 155] [3, p. 216] has originally been designed for locating roots of equations, but it can be used for function optimization. The method uses a Hessian matrix which contains the second order partial derivatives and describes the local curvature of the function. The matrix is used in generating a quadratic approximation of the function to be optimized, which matches the first and second order derivatives of the initial point. This approximation is then used to decide the direction and length of the local search step. If the local neighborhood of the optimum forms a quadratic shape, the algorithm jumps directly to the optimum. Newton's method is typically considerably faster for functions which offer reliable second order derivative information, i.e. allow the Hessian matrix to be reliably generated compared to CG methods. Of course the use of the second order derivative also limits severely the set of functions for which Newton's method is applicable.

Quasi-Newton (or variable metric) methods [26, p. 187] [121, p. 425] aim to extend the usability of Newton's method by building an estimate of the Hessian matrix iteratively and using the current estimate at each iteration in conjunction with Newton's method. The Hessian is approximated by analyzing the history of gradient vectors, and as a consequence, quasi-Newton methods do not require the second order derivatives. However, they require storing the accumulated information, which is of a higher order of magnitude than CG methods. Several slightly different variations of the general Quasi-Newton concept exist [26, 121].

2.2.2 Direct search methods

The *Nelder-Mead simplex* (or downhill simplex) algorithm [109] [121, p. 408] uses a simplex, which is a geometrical figure consisting of $D+1$ points called vertices, D referring to the number of dimensions of the search space. In two-dimensional cases, the shape of the simplex is a triangle. Unlike most LS methods, the Nelder-Mead simplex requires not one, but $D+1$ starting points to form the initial simplex. The idea is to use the vertices of the simplex to estimate moves that increase the fitness and replace the worst by the improved point. A reflection step takes the worst vertice and reflects it to the opposite side of the simplex in the hope of finding an improved solution. An expansion step, where the reflection step length is increased, is performed if the reflected point is better than any of the current vertices to exploit a promising direction. The expanded point is then used instead of the reflected point if it has better fitness to replace the worst vertice. If the expanded point does not improve the fitness value, a contraction step is taken, where the worst point is drawn towards the centroid of the simplex. If the contracted point is able to improve the fitness, it replaces the worst vertice; otherwise all vertices are moved towards the best vertice, resulting in a smaller but similarly shaped simplex. Through these steps, the simplex is able to adapt itself to the function to be optimized and converge to a local minimum.

Pattern search procedures search the neighborhood of an initial point according to a predetermined pattern. The *Hooke-Jeeves* [67] algorithm is a pattern search method which decides the search direction through explorative step attempts for each dimension separately. An explorative move is performed by copying the current base point (initial point is the first base point) first to $\vec{\gamma}$. A predetermined positive number determining the step length is then added for the first parameter of the $\vec{\gamma}$ and the fitness is calculated for the resulting point. If the fitness is improved, the new point replaces $\vec{\gamma}$. Otherwise a similar step is attempted in the opposite direction. If neither one of the step attempts improves the fitness, $\vec{\gamma}$ remains unaltered. Similar explorative axial moves are attempted for all D dimensions, and then the $\vec{\gamma}$ is set as a new base point. A pattern move is then performed by adding the differential between the two latest base points to the newest base point. This point is added as a new base point if it improves the fitness. If all the axial move attempts fail, i.e. the new base point is identical to the previous one, the step length of the axial moves is reduced. The procedure is then repeated. Minimum step length can be used as the end condition for the search.

Another pattern search procedure, the *Rosenbrock* method [129] similarly starts the search in axial directions. Search steps for each parameter are performed in turn, so that if the previous step in that direction increases the fitness, the step length for the next jump for the same parameter is increased. Respectively, for a failed step attempt, the step length is decreased and the direction is set to the opposite. The procedure is repeated until the algorithm has recorded both successful and failed steps in all possible $2D$ search directions. At this point, the coordinate system is rotated in regard to the direction from the initial point to the final point, and the process is repeated using the rotated axial directions. The rotated direction approximates the gradient and allows the algorithm to adapt to the shape of the optimum.

A simple way of acquiring LS-like behavior is to simply generate a sample of points in the neighborhood of a point, to select the best point, and repeat the process. The term *stochastic local search* (SLS) [68] is used in this work when referring to methods which use a specific distribution to generate the set of points for the LS randomly. Generally, however, the term is often used for referring to a wide variety of algorithms based on random sampling, many of which are not limited purely to local search. The classification for stochastic methods in general to LS and GO methods is difficult, because increasing the area in which the random points are generated allows such algorithms to jump to different optima (and decreases the local search capacity). When the area in which the points are generated is small compared to the whole search space, such algorithms work as local searchers. At the other extreme is pure random search, where random points are generated for the whole search space. For this reason, the term *extended local search* is used in this thesis to refer to methods or operators which can not be strictly classified to belong to either LS or GO groups. Methods doing extended local search concentrate their efforts on a part of the search space, inside which they aim to locate the best optima. They are able to escape local optima locally, but do not consider the whole search space at once, like GO methods. *Crossover-based local search* (XLS) operators [91] are extended local search operators used in the context of evolutionary algorithms, which acquire the local sample points by performing crossover operation between population members.

2.3 Local search-based global optimization methods

The most straightforward way of implementing a GO method which contains both the local and global search phases, is to keep restarting a local search algorithm from different starting points and storing the best found optima until an end condition is reached. Such strategies are referred to as *multistart methods.* The success of such methods depends directly on the selection of the initial points. It is not trivial how the points should be selected to achieve good exploration power in a general case. A simple solution is to select the starting points using a pure global search method, like grid search or pure random search, and use the generated points as starting points for a local search procedure. Such methods retain the feature of being theoretically convergent but are more useful in practice, compared to pure random search, due to their ability to locate optima faster.

In this work, two multistart approaches are used, both of which use the gradient descent as the local search method adopting the line search procedure described in Section 2.2.1. *Random start gradient descent* (RSGD) uses uniform random starting points, while *grid-based gradient descent* (GRGD) uses a predetermined grid for generating the starting points. The gradient vector is always approximated with a central difference estimate, using the largest absolute value of a box constraint for any of the dimensions of the problem, scaled with $\eta = 10^{-8}$ as the distances from the central point.

A grid for GRGD is generated to maximize coverage. First a single point is generated at the middle of the search space in regard to all dimensions. Then, new points are generated between existing points and the box constraints at each subsequent generation such that all distances are halved. Finally any remaining gaps are filled, such that the acquired points are evenly distributed in the form of a hypercube. The number of points generated at the end of generation g is $(\sum_{i=0}^{g} 2^i)^D$. As can be seen, each new generation will contain more starting points than all previous the generations combined.

Especially with higher dimensions, the number of generations completed will be low, and the search typically stops short of completing a generation. Using the starting points in a predetermined order would create a strong bias in the search. For this reason, the points inside each generation are used in a random order to minimize the bias. Of course, bias can not be completely eliminated from the grid-based approach. The chosen method of generating grid makes the algorithm inefficient in searching optima located on the constraints. Generating starting points on the constraints at the beginning would improve the performance on locating optima which are at the constraints. Such a strategy would be problematic in a more general setup, however, because it would strongly bias the search to the constraints, as 3^D points would be required to achieve an initial coverage of the constraints. While this works in low-dimensional cases, in problems with higher dimensionality the algorithm would only generate points on the constraints. Algorithm 1 presents the RSGD and GRGD algorithms.

Algorithm 1 Gradient descent algorithm

```
1: while termination criterion not met do
2:   Pick $\vec{x}_a$ randomly (RSGD) or using grid (GRGD)
3:   while the optimum has not been found with required precision do
4:     Calculate normalized gradient $\vec{g}_n = \vec{g}/|\vec{g}|$
5:     Perform bracketing using method presented in [121, p. 400], using initial points
       $\vec{x}_a$ and $\vec{x}_b = -s \cdot \vec{g}_n + \vec{x}_a$
6:     Perform line search using method presented in [121, p. 404]
7:   end while
8: end while
```

One drawback of the multistart methods is the fact that they tend to locate the same optima repeatedly. The effort allocated for locating an already found optimum is wasted. *Clustering methods* aim to select the starting points for a multistart method so that the number of wasted local searches is minimized. Schoen [141] defines a generic clustering framework for generating the starting points for a multistart method. The process starts from a uniformly distributed set of points. Then a sample concentration is performed to change the distribution of points to reflect the shape of the function. Sample concentration can be done for example by simply discarding a predefined percentage of the points with worst fitness. Another typical method is to run an LS algorithm a few steps for each point of the sample. A clustering method is then used for the concentrated sample dividing the points into clusters, which estimate the locations of different local optima. The LS procedure is then run for only one representative point from each cluster, as it is assumed that points belonging to the same cluster belong to the area of attraction (AOA) of the same optimum, i.e. LS started from any point would lead to the same optimum. The procedure can be repeated several times if necessary. A wide variety of different clustering methods have been suggested for the general framework. For more information see for example [141, 70, 71, 167].

The multistart methods presented above aim at locating the global optimum by finding a way to place the starting point of the LS method inside the AOA of the global optimum. Another approach for implementing the multistart approach is to modify the function between restarts in such a way that the original starting point has only little impact on the final solution. One way to perform the modifications is to use *direct elimination of* *local optima* [108]. The idea is to simply eliminate each found optimum from the search space by modifying the function locally, so that the next iteration will not end up in the same optimum. The main difficulty of such methods is determining the size of the neighborhood to be eliminated: choosing too small a neighborhood will create new false local optima, while too large neighborhood value may remove other real optima.

Local smoothing refers to methods which modify the original function by smoothing the function to get rid of poor local optima. The idea is to start from a very smooth function, whose optimum can be easily located, and gradually decrease the degree of smoothness. The solution from the previous round is used as the initial point for the next round. Moré and Wu [105] for example use a Gaussian transform for generating the smoothed approximations of the function. A single numeric parameter controls the level of smoothing. The limitation of local smoothing is that the approximation does not necessarily correspond to the original function and can draw the search towards a local optimum instead of the global one. The Gaussian transform biases the search towards the middle of the search space, which may lead the method to ignore the global optimum close to the box constraints even in very simple functions [108]. For this reason, local smoothing is most useful for removing local noise.

Global smoothing adds a convex function to the original problem, and the fitness is calculated as a sum of these. The idea resembles local smoothing, and the aim is to similarly eliminate poor local optima. A multiplier term defines the smoothing effect the convex function has to the original function, and similarly to local smoothing, the optimization starts with a high value and decreases between the runs. The main difference between local and global smoothing is that global smoothing guarantees the function to become unimodal with a large enough multiplier. Local smoothing methods can not make such a guarantee. The initial point for the LS has no effect on the outcome of the search for global smoothing. The optimum location of the added convex function decides the outcome of the search: if it is close to an existing optimum of the original function, it will direct the search that way, although the optimum may not be global. For this reason the task of finding good initial points for LS is transformed to finding good optimum locations for the added convex function.

Shang [144] uses a space filling curve to explore the search space, coupled with an LS method. The fitness is calculated by taking into account the distance to the space filling curve in addition to the actual fitness of the optimized function. The search thus only deviates from the curve when a good attractor is spotted, whose pulling force is able to overcome the penalty of straying away from the curve. The penalty has a smoothing effect to the fitness, like local smoothing. An obvious difficulty with the method is defining a suitable penalty for straying from the curve. Another problem is that the used space filling curve biases the search in the same way as using a grid to decide the starting points of LS if the whole curve is not searched. Space filling curves will become very long in higher-dimensional problems.

2.4 Metaheuristic methods

Metaheuristic methods are stochastic and heuristic (do not guarantee exact solution) general global optimization algorithms using *black-box* procedures. The term black-box refers to the fact that the algorithm does not make any assumptions on the optimized problem. The only required information is the fitness value with a given input. As the metaheuristic algorithms are general optimizers, they are rarely the most efficient methods for problems for which an efficient problem-specific algorithm exists. Thus metaheuristic methods are best used for problems for which problem-specific algorithms are too hard or even impossible to construct. Typically metaheuristic algorithms require a set of user-specified parameters, whose values determine the success of the algorithm in the given task. Thus the user may provide problem-specific information through the selection of suitable parameters. On the other hand, if such information is not readily available, finding a good parameter setup may become a difficult task by itself. Often metaheuristic methods are further tailored for a specific problem by taking a general framework and importing some problem specific-data to the algorithm through modifications.

To be able to solve a black-box problem, metaheuristic algorithms need to be able to capture and exploit problem-specific characteristics. To do this, some form of a generational memory is required, which stores important information and allows it to be used to guide the search. This can be a memory structure which stores information of previously visited points, or a population, which simultaneously contains a set of solutions for the problem. The reasoning on using the population is that it can capture useful information of the problem at hand, which can be used to direct the search on a global scale. This section introduces well known metaheuristic approaches.

2.4.1 Non-population based methods

Simulated annealing (SA) [76, 90] was first introduced for combinatorial optimization, but has been later generalized for real coded functions. The algorithm simulates the cooling of molecules to a state of minimum energy (optimum). The general framework of the algorithm requires the definition of three main parts: distribution for sampling new points, acceptance function, and cooling schedule. The idea is to start from a random initial point (state) and generate a new random point according to the sampling distribution. The acceptance function is then used to decide whether the algorithm moves to the new state or not. To allow the algorithm to escape local optima, the acceptance function typically allows moves to worse points sometimes. The cooling schedule controls the probability, so that at the beginning, moves to worse points are more likely, and the probability decreases as the system cools down and the algorithm converges. The basic idea of SA falls to the category of extended local search, as it actually performs stochastic local search in the neighborhood of the current state. The global information is limited to the cooling schedule, as the framework does not implement a generational memory. The framework, however, leaves the three main parts open, and variations using the history of sampled points to direct the search have been presented [90]. For example, some simulated annealing methods try to scale the probability distribution based on the previously visited points.

Tabu (or taboo) *search* (TS) [51, 52, 53] was also initially designed for combinatorial problems, and later extended for real coded functions [29]. Similarly to SA, it uses a single point, which moves around the search space. Different methods for generating the sequence of points can be used, and usually LS methods are used as a part of the approach. The essential idea behind TS is the use of tabu lists, which are used as the generational memory to direct the search globally. Tabu lists define some areas of the search space as taboo and direct the search away from these areas. Different rules for the lists can be used. Typically areas around the already found solutions are excluded to prevent the algorithm from searching the same areas multiple times. Essentially, the algorithm implements an advanced "direct elimination of optima"-principle, as the taboo lists are often temporary and also exceptions can be specified.

2.4.2 Evolutionary algorithms

Evolutionary algorithms [40, 17, 18] are a class of metaheuristics, which aim to use the concepts of natural evolution in optimization. The underlying idea of EAs is to use a population of candidate solutions which evolves through genetic variations and natural selection. The selection realizes the "survival of the fittest" principle: population members with higher fitness are more likely to survive and participate in generating offspring. This creates a convergence pressure for the population towards better solutions. New candidate solutions are created through stochastic *variation operators*, which use the population information to direct the search to promising regions. The selection and variation operators are the two fundamental forces driving the evolution process. The selection procedure is often seen as being exploitative by nature, while the variation operators are responsible for exploring the search space. Such a straightforward connection may be misleading, however, as pointed out by Eiben and Schippers [39]. In this thesis the term exploitation is used in the context of defining the ability of an algorithm to exploit specific function features. Such abilities are rarely related to the selection procedure, but rather to the used variation operations.

Representation

In order to solve a problem by using an EA, the representation must be decided upon. The term *phenotype* refers to the actual candidate solutions of the optimization problem, while *genotype* refers to the representation of population members which correspond to the phenotypes. While sometimes EAs work directly using genotypes like real coded EAs solving real number problems, the phenotype must often be encoded to form a genotype. The used codings are an important topic especially in the context of combinatorial optimization, but as this thesis is limited to continuous optimization, the topic is not considered further. An exception is *binary coding*, which is widely used in solving different types of problems including functions with real parameter spaces. The idea is to simply represent each parameter of the solution by a binary number of a fixed length. These numbers are often called genes, and a chromosome (individual) is formed by combining these genes. When real numbers are encoded to binary coding, the length of the gene decides the achievable precision, as the encoding is done from a continuous to a discrete set. *Gray coding* [21] is sometimes used instead of raw binary numbers to avoid the forming of Hamming cliffs.

Variation operators

Variation operators are typically classified to *mutation* and *recombination* (crossover) operators. The defining feature of mutation operations is that they are unary operators, i.e.

they are applied to a single population member. Heuristic unary operators are typically not classified as mutation, as a mutation operator should cause random changes which are not biased by any predetermined decisions. The population information, however, is often used for biasing the mutation distribution. In binary coded representation, mutation is typically done by simply flipping random bits of a chromosome. For real coding, the mutation typically generates a random vector from a suitable distribution and adds that to a population member. The role of mutation is to allow access in completely new areas of the search space, but mutation can also work as an SLS operator, if small step length is used. *Self-adaptive* methods are able to adapt some of their parameters, like the mutation step length or the shape of mutation distribution, during the optimization process. Thus the same mutation operator may perform longer, explorative jumps at the beginning and then become an SLS operator when the jump length decreases.

The idea behind recombination is to merge information from two or more population members. The reasoning behind the use of recombination is the building-block hypothesis [65], which assumes that better solutions can be constructed by combining good partial solutions. The idea is simply to take parts from two already good parents in the hope of generating even better solutions. *One-point crossover* is the simplest of the recombination operators. The idea is to simply split two parent individuals from a randomly selected location and switch the tails. The process creates two offspring. The *n-point crossover* is a generalization of the one-point crossover. Now instead of splitting the parents to half, they are split to n parts, and offspring are created by taking alternative parts from both parents. *Uniform crossover* treats each variable independently, using a predefined probability when deciding which parent it is inherited from. All the listed methods can be applied to both binary and real coded representations. However, the splitting for real coded representation can be done only between the variables and not between bits, as in the binary coded version. This means that the recombination for real coding can only create new combinations of existing variables, i.e. the created points are biased to the axial directions and allow only a fixed number of different possible jumps. For this reason *arithmetic* (intermediate) *recombination* is often used with real coding. The basic idea is to create the offspring between the parents through an averaging process. Such averaging operators create a bias towards the center of the population. This is often not desirable, provided that no problem specific information is available to indicate that the search should concentrate on the middle of the search space rather than the borders. Due to the difficulties of applying recombination, real coded algorithms tend to use mutation as their main operator, while binary coded algorithms typically rely mainly on recombination.

Selection operators

Two types of selection are used in EAs: *parent* (or mating) *selection* and *survivor* (or environmental) *selection.* Both types of selection are always present in any EA, but often an EA emphasizes one and the other is implemented in a straightforward way. Parent selection is performed prior to the variation operators and decides which of the population members participate in the breeding process. Survivor selection is performed after the variation operators are applied and decides which of the created offspring are granted entry to the population.

EAs can be classified to two different categories on the basis of how the selection operators are divided into periods: *generational* and *steady state* algorithms. In generational algorithms, a large pool of offspring is first created through parent selection and variation operations. Then survivor selection is applied, and the whole population is replaced at once. Often generational algorithms emphasize parent selection and survivor selection simply discards the old population and replaces it with the population of the offspring. An important concept, especially with generational algorithms, is *elitism*. An algorithm is elitist, if it guarantees to never lose the best solution found thus far. Generational algorithms are typically non-elitist if special precaution is not taken to ensure the survival of the best solution between generations. In contrast, steady state algorithms use survival selection to replace only a part of the population at a time. Typically steady state algorithms emphasize the survival selection and use the fitness or sometimes age information to decide if the offspring are allowed to enter the population and which population members are replaced.

The simplest way of implementing parent selection is to simply allocate the population members as parents randomly without considering their fitness. Another method is to allocate each population member in turn as a parent in a predetermined order. A more sophisticated selection method is *fitness proportional selection* (FPS). The selection works so that the probability of each individual to be selected for breeding is directly proportional to its absolute fitness. FPS constitutes three main problems: individuals that are significantly better compared to the rest of the population will be selected for breeding very often, and as a consequence will fill the population with their offspring very quickly, which can lead to *premature convergence*, i.e. convergence to a local optimum. On the other hand, if the fitness values of the population are almost equal, FPS degrades to uniform random selection constituting no real selection pressure. Finally, the behavior of FPS changes if the coordinate values of the function are shifted. *Ranking selection* (RS) has been developed to fix the problems of FPS. The idea is to use ranked fitness instead of absolute fitness. Individuals are first sorted according to their fitness, and each is then allocated a predetermined probability for being selected for breeding, based on their rank. *Tournament selection* differs from FPS and RS in the way that it does not consider the whole population at once, but rather picks a sample of n random population members and arranges a tournament between these. Depending on the implementation, the winner is either directly the individual with the highest fitness, or it is randomly selected, by giving a higher probability to the ones with better fitness. The selection pressure can be controlled through tournament size n. FPS, RS and tournament selection can be used in the survivor selection phase as well by combining the current population members and offspring. Alternately, some methods use *deterministic selection* by simply combining the parent and offspring populations and keeping the best n solutions.

Classification

Historically, EAs suitable for continuous optimization are classified into three main categories: genetic algorithms (GA) [65], evolution strategies (ES) [9, 142] and evolutionary programming (EP) [43, 44]. Genetic algorithms are the most well known, and they are characterized by the use of binary coding and recombination as the main operator. ES and EP, on the other hand, have initially been designed for real coding and use mutation based on normally distributed random numbers as the main operator. Both typically use self-adaptation by including some of the control parameters to the population vectors, and they evolve along with the solutions. The main difference between ES and EP is that ES allows the use of recombination while EP methods are purely mutation-based. Additionally, ES conceptually uses deterministic survivor selection while EP applies tournament selection. Several real coded variants of GA have also been presented [63]. Such methods, however, often resemble ES or EP more than binary coded GA. As the classification to GA, ES and EP is based on history and not directly on the properties of algorithms, it is becoming increasingly difficult to define a certain method to belong to a particular category, especially because the methods of each group have been further developed and have adopted ideas from each other. Additionally, some EA methods, such as Differential Evolution [125], do not fit well to any of the three categories.

Deb [34] presents an interesting idea of a population-based algorithm generator by defining four basic plans which define an optimization algorithm. The selection plan describes the parent selection method. The generational plan describes how the selected parents are used to generate offspring. The replacement plan describes which solutions of the population will be subjects to replacement, and the update plan is used to decide which individuals among the offspring, parents and the solutions selected for replacement will actually enter the population. Deb demonstrates that such a generator is able to represent different types of EAs, as well as other types of optimization algorithms, and generate new algorithms by using different combinations of the plans.

2.4.3 Other approaches

In addition to EAs, another group of metaheuristics, which takes its inspiration from biology, are the methods based on swarm intelligence [75]. Among these, *particle swarm optimization* (PSO) [74] is best suited for continuous optimization. The method models the swarm behavior of animals or insects. Population members are referred to as particles, which fly through the search space and evaluate the fitness values of visited points. The direction and velocity (step size) of each particle is affected by two attractors: the best location the particle has personally visited this far, and the best known point located by any particle in a certain neighborhood. *Gbest PSO* methods refer to variants for which the neighborhood is defined as the whole search space, while *lbest PSO* refers to methods for which the neighborhood is defined in a more limited manner.

Harmony search [49, 82] mimics the inspiration of music players to search for the perfect state of harmony. Harmony memory (HM), which imitates a population and acts as the generational memory, is initialized with random values. New harmonies are improvised by using a procedure resembling recombination. The value for each variable is independently chosen from a member residing in the HM, or generated randomly. The probability of either is controlled through a predetermined control parameter. If the variable was taken from HM, it may be pitch-adjusted. Pitch adjusting simply adds a small value to the variable providing SLS to the algorithm, comparable to a mutation operator in an EA. The created harmony replaces the worst harmony in the HM, if it was able to improve the fitness.

2.5 Hybrid methods

Hybrid methods aim to combine different optimization approaches in a way that allows the hybrid to employ the strengths of parent algorithms and to minimize their weaknesses. In principle any combination of algorithms can be hybridized, but usually the best results are achieved by combining algorithms which can offer different strengths to the hybrid. Especially interesting are combinations of local and global optimizers. The term *memetic algorithms* [61] refers to methods that combine EA with local search methods in an effort to increase the local convergence speed without losing the advantages of adaptive generational memory. Pradeeb and Ranjan [120] divide such methods in three main categories according to when the local search is applied: pre-, post- and organic hybridization.

Pre-hybridization uses a local optimizer for generating the initial population for EA. The methods are mainly useful for directing the search to user-defined areas and increasing the convergence speed (compares to seeding the population with already known solutions). While such approaches may offer increased performance in specific cases, like in [83], they generally tend to decrease the diversity of the initial population and thus hinder the exploration capabilities of the EA part, increasing the risk of premature convergence.

Post-hybridization methods run EA first to achieve a starting setup for a local search. The aim is to achieve good exploration of the search space by first running an EA and then switching to local search to speed up the final convergence. The main difficulty in such approaches is determining when to perform the switch. For example a method presented by Chelouah and Sierra [24], starts with GA and switches to a Nelder-Mead simplex when the progress of GA starts to fall, or a fixed number of generations is reached. The hybrid genetic-conjugate gradient algorithm presented by Pradeep and Ranjan [120] runs GA multiple times and uses the best points obtained for the starting points to conjugate gradient local search.

Organic hybridization methods use local search to improve the solutions during the EA run. A typical approach is to use local search to improve the individuals produced by the EA. A lot of research has been done on organic hybridization of GA and a good outlook of the field, as well as an example of one such approach, can be found in [91]. Organic hybridization methods are historically divided to Lamarckian and Baldwinian ones [177]. Lamarckian methods directly use the individual improved by the local search, while Baldwinian methods attach the fitness value of the improved individual to the original produced by the EA. For this reason, Baldwinian methods are able to keep up the diversity longer, leaving more time for the evolutionary algorithm to explore the search space. On the other hand, Lamarckian methods tend to converge faster, but typically include a higher risk of premature convergence.

2.6 Summary

This chapter has presented an overview of continuous global optimization. GO methods consider the whole search space and aim at locating the global optimum. Pure GO methods do not contain any local improvement phase, which induces slow convergence properties, unless problem-specific information is used to direct the search. Local search methods, on the other hand, aim at locating an optimum residing near the starting point of search efficiently without worrying whether the found optimum is global or only local. Derivative-based LS methods are efficient, but require at least numerical approximation of the derivative to be available. Direct search methods are typically less efficient, but do not require derivative information and are thus applicable to a wider selection of problems. LS methods can be used for global optimization by using a pure GO method for generating starting points for the LS method. Such multistart methods are able to improve the convergence speed of pure GO methods. Another way of implementing multistart methods is to modify the function between restarts so that the initial starting point has only little impact on the final solution.

Metaheuristic algorithms are general black-box GO algorithms. They often draw inspiration from natural or physical processes, such as evolution. They aim to capture and exploit problem characteristics through a generational memory, which is often implemented as a population of solution candidates. Hybrid methods combine different algorithms to employ their strengths and minimize the weaknesses. Especially combinations of metaheuristics and LS methods offer a potential to achieve algorithms with superior cababilities in many optimization problems.

This chapter has concentrated on analyzing methods for locating a single solution for a given optimization problem. In the next chapter, methods for locating multiple solutions simultaneously are considered.

Chapter III

Multimodal global optimization by niching

An optimization problem is considered to be multimodal, when it contains more than one locally optimal solution. The goal of global optimization is typically to find the solution having the best fitness value among the optima - the global optimum. Of course, a multimodal problem may also contain multiple globally optimal solutions. Examples of real world problems with multiple global optima include locating the DC operating points of a nonlinear electronic circuit [28], or power system planning [38].

In multimodal global optimization, the goal is to locate several optima. Typically the task is to locate at least all global optima. However, which of the optima are considered global depends in practice on the used precision. Decreased precision may make local optima, which have a fitness value close to the global optimum, indistinguishable from the global optimum. Additionally, some of the locally optimal solutions, while being inferior to the global optimum in terms of their fitness value, may provide other interesting qualities which are not included in the model being optimized. From a practical viewpoint, multimodal optimization gives the decision maker options to choose from by offering multiple possible solutions. Thus the goal of multimodal global optimization may be defined as locating the best n optima of a multimodal problem, so that n is not smaller than the number of global optima (NGO) of the problem. The best optima may be defined purely by the fitness function value. Typically, however, also a distance component is taken into account: in a rugged landscape, local optima near a global optimum are often not interesting, even though they typically have good fitness value. More interesting is locating the *locally global optima*: solutions that are optimal in promising regions of the search space.

3.1 Niching

The term niching refers to natural ecosystems, where different species survive by evolving to fill different niches. These niches can be seen as stable subpopulations, each covering a single promising region of the search space. Niching methods can be broadly classified into two categories: sequential niching (SN) and parallel niching (PN) methods [96]. Especially the parallel niching methods are closely related to population-based optimization methods, like evolutionary algorithms. Sequential niching methods rely on running an optimizer repeatedly, and do not necessarily require a population.

The main purpose of niching is to allow an algorithm to locate and maintain multiple solutions. In algorithms using population, this typically means maintaining the population diversity, i.e. preventing or at least significantly slowing the population from converging to a single region of the search space. This potentially decreases the chance of premature convergence, and thus niching methods are often used even when the goal is only to locate a single global optimum. In the context of this thesis, however, the most interesting feature of the niching methods is their ability to allow optimizers to perform effective multimodal optimization.

3.2 Complexity

It is customary to define the computational complexity of niching methods on generational basis [104, 140, 46, 84, 27, 85, 151, 113]. In one generation (the term iteration is often used interchangeably), a population-based optimizer typically produces NP new individuals (trials), i.e. searches for NP new locations of the search space, which is usually also the number of required fitness function evaluations. Niching methods do not typically require excess function evaluations, and complexity increase comes through the calculations required to decide the similarity of population members. The generational complexity for a niching method is defined so that if the method does not add additional computational complexity on top of the complexity of an optimizer (like preselection), the complexity is defined to be $O(NP)$, based on the number of generated trials in one generation. For a method which requires a similarity measurement calculation between each trial and each existing population member (like sharing), complexity is defined as $O(NP^2)$, because the total number of required similarity measurements in a single generation is NP^2.

While generation-based complexity measurements give a measure of proportionate complexity between most niching methods, they are limited to algorithms for which the concept of generation can be applied. They can also be misleading: defining complexity as $O(NP)$ suggests that the complexity of the niching should increase with the population. However, increasing the population size also increases the number of trials, and as a result the number of fitness function evaluations performed inside one generation. Thus the complexity per generated trial stays constant. The progress of the search of an algorithm is proportional to the number of generated trials, not to the definition of a generation, and the complexity measure should reflect that.

Defining the average added complexity of the niching method for a single trial gives a more useful measurement of the complexity of different niching methods. This also allows direct comparison to non-generational methods. Using the *complexity per trial* (CPT) measure, the complexity of preselection is $O(1)$ and the CPT of sharing is $O(NP)$. Now it is instantly clear that the complexity of preselection is independent of the population size, and the complexity of sharing requires NP extra operations for each generated trial, i.e. each search step the algorithm makes. A similar representation for complexity has been used by Menczer and Belew [98]. CPT is used for the rest of this thesis as a default complexity measure of niching methods.

3.3 Sequential niching

Multimodal optimization can be performed simply by running any optimization algorithm multiple times with varying initial configuration and recording the found optima. The problem of such approaches is that they typically waste a lot of time by searching the same regions over and over again. For local optimizers, various methods have been suggested for minimizing excessive search on regions already visited, like constructing the starting points in a grid or using clustering to eliminate processes which seem to be heading towards the same optimum. The aim of SN methods is to use the information from previous steps to guide the search away from already explored areas. They may modify the optimization landscape by removing already found attractors, for example using direct elimination or local smoothing techniques. Another approach is to add additional constraints to prevent access to certain regions of the search space. Tabu search, for example, memorizes the already visited locations and uses the information to direct the search to more promising areas.

The derating technique presented by Beasley et al. [7] runs GA repeatedly and records the best found solution for each run. The derating function is used to penalize the fitness value around the already found optima within a niching radius. Vitela and Castanos [172] have modified the original method by extending the effect of the derating function outside the niche radius and using a clearing procedure to eliminate solutions within the radius. The *adaptive isolation model* of Ando et al. [2] uses deration in conjunction with clustering during a GA run to isolate subpopulations. The procedure is applied recursively for the subpopulations. Zhang et al. [187] use the derating concept in conjunction with particle swarm optimization to achieve multimodal optimization. A related method proposed by Parsapoulos and Vrathis [114] modifies the search space during a particle swarm run. The aim is to remove the tendency of the swarm to oscillate between several attractors by isolating and searching them independently by a small subswarm.

Mahfoud [96, 94] has compared niching methods in combination with GA, using the derating technique as an example of sequential niching. His experiments demonstrate a better performance for PN both in terms of speed and number of located optima, and he concludes that PN outperforms SN for the following reasons: deration is unable to completely eliminate the problem of locating the same solutions repeatedly, but it can transplant or destroy optima of interest or create false optima around the already found solutions. These effects increase the difficulty of locating the remaining optima, as the problem turns to "multiple needles in a haystack" [94, p. 209]. Derating may thus actually make the problem harder and not easier, as was the original idea. Additionally, PN allows parallelization and can be used to maintain diversity also when locating only a single solution, unlike SN. While there may be ways around some of the shortcomings of SN, the attention for the remaining of this thesis is restricted to parallel niching techniques, which are seen to hold greater potential for multimodal optimization.

3.4 Parallel niching

A population forms a natural foundation for multimodal optimization algorithms, because it allows multiple solutions to exist simultaneously. Population-based global optimizers in their basic form, however, are typically designed to eventually converge towards a single solution, i.e. they will eventually lose population diversity. Parallel niching methods aim to prevent this. To understand the concept of PN, the concept of population diversity must be understood first. According to Mahfoud, "Diversity is a general term that describes variation or lack of similarity among a collection of objects" [94, p. 51]. Depending on the coding of the population, several numerical measures may be used for estimating the population diversity. For example the sum of Euclidean distances between all real valued population members is one such estimate. As numerical diversity measures are not used in this thesis, their exact definitions are outside its scope. A general framework for defining diversity measures and several examples can be found in [94]. Regardless of the exact measurement used, the diversity is high when the population is evenly spread throughout the search space and diminishes when population members start to form groups around good solutions and is at minimum level when the whole population has converged to a single point. In population-based optimizers, initialization is often done by using uniformly distributed random numbers over the search space. Thus population diversity is typically at maximum level at the start of the optimization process and diminishes as the population converges.

Simply maintaining a high level of population diversity is not itself the goal of an effective PN method. As the diversity is high when the population members are just random points, an effective way of keeping the diversity would be to simply prevent the optimizer from converging. Of course this kind of an optimizer would be useless, as it would not be able to locate optima effectively. Thus the goal of a PN method is to maintain useful diversity, not to completely prevent convergence. Mahfoud states that "... diversity is useful if it helps in achieving some purpose or goal" [94, p. 60]. In case of multimodal optimization this means that the niching method must allow convergence locally to the desired solutions, but at the same time be able to maintain diversity between different regions of the search space.

3.4.1 Implicit parallel niching

Zaharie [183] divides parallel niching methods to implicit and explicit ones. Methods using implicit PN work with a single population. Niching is achieved by modifying the optimizer itself to favor solutions that keep the population diverse. Explicit PN, on the other hand, divides the population into subpopulations explicitly and runs the optimizer for each of these independently. Implicit PN methods are the most well known group of niching methods. Most of the methods have originally been developed to work in the context of genetic algorithms and later generalized for other population-based optimizers.

Restricted replacement

One way of achieving implicit PN is through restricted replacement. The basic idea is to allow competition only between similar population members, which can be seen as belonging to the same niche. One of the earliest studies of restricted replacement in the context of GAs are the three *preselection* schemes by Cavicchio [22]. Scheme 3 demonstrated superior performance compared to the other two. It pits an offspring against its weaker parent, decided by the fitness value. The offspring replaces the parent, if it has a superior fitness value. The reasoning is that the offspring inherits a lot of features from the parents. Thus the differences between the parents and the offspring are typically smaller than between the offspring and the other population members. When the offspring replaces a parent, which is one of the most similar population members, the population will not lose a lot of useful diversity, but still allows the population to converge locally. The main advantages of preselection are low CPT of $O(1)$ and the lack of need to define control parameters, or even a formal measure for similarity between the population members.

De Jong has introduced the *crowding* scheme in his thesis [33]. The concept was initially designed for maintaining population diversity to prevent premature convergence, not to actually enable multimodal optimization. In crowding, an offspring is compared to a random sample taken from the current population, and the most similar individual in the sample is replaced. Crowding factor parameter CF is used to determine the size of the sample. To define the similarity, a difference measurement must be defined. Typically the difference is seen as a distance between two population members. De Jong measured the distance by using genotypic matching (Hamming distance), by performing a bitwise comparison between the population members. The problem with this measure is the fact that it does not distinguish between more and less significant bits. A more recent distance metric for binary strings, phenotypic similarity [56], decodes the parameters before comparison to overcome this. A typical similarity measure between real coded population members is the Euclidean distance.

Harik [60] suggests a *restricted tournament selection*, which modifies tournament selection to achieve niching cababilities. The method resembles De Jong's crowding. Two individuals are first selected from the population to produce two offspring through mutation and crossover. A binary tournament is then run between each offspring and the most similar individual from a random sample of individuals taken from the population. Restricted tournament selection also requires a user to specify the sample size equal to the CF parameter of De Jong's crowding.

Mahfoud [95, 94] has examined both crowding and preselection. He defines a *replacement error* (RE) to happen, when a population member residing inside the AOA of one optimum gets replaced by a member from another. While this definition fits for problems where the goal is to locate all the optima, a more general definition is required when the goal is to locate only n best optima among a large number of optima. For such cases, the ability of the algorithm to lose unwanted optima is essential for success, and an algorithm achieving zero replacement error according to Mahfoud's definition will achieve a poor performance, because population members will be stuck in the initial optima. This will be demonstrated in the experimental part of this thesis. *Unwanted replacement error* (URE) is defined to happen when a trial replaces a population member which is not the most similar according to the used similarity measure.

Mahfoud's results demonstrate that while using small values for CF, replacement errors decrease the ability of crowding to maintain multiple optima. Increasing CF decreases the frequency of REs up to value $CF = NP$, which eliminates the UREs. The downside of this is increased computational complexity: as CF distance measurements are required between each trial solution and the population, the CPT of crowding is $O(CF)$. UREs are an issue also with preselection, due to the fact that the algorithm always replaces the worse parent: because an offspring may inherit a varying number of features from the parents, a parent contributing less may be very dissimilar to the offspring. If this parent has a worse fitness value compared to the other parent, it is likely that the offspring, which resembles the better parent, will also have superior fitness. Replacing the worse parent thus often causes UREs.

Based on his findings, Mahfoud suggests a *deterministic crowding* [95] method, which aims to overcome the replacement error problem of preselection by using the idea of similarity measurements from crowding to decide which of the parents is closer to the offspring. The algorithm divides the population randomly into $NP/2$ pairs at each generation. Each pair undergoes crossover and mutation to produce two offspring. Now one of the children competes against one parent and the other child competes against the remaining parent, the survivor being decided by the fitness and a tie favoring the parent. The order is decided by minimizing the combined phenotypic similarity between both child-parent groups. This fixes the main RE problem of preselection, because now the offspring will compete against the more similar parent. This virtually eliminated the REs in Mahfoud's test cases. However, deterministic crowding can not completely eliminate UREs (like crowding using $CF = NP$), because the population may contain even more similar individuals compared to the most similar parent. As deterministic crowding requires only two distance measurements for each trial solution, it shares the advantages of preselection, the low CPT of $O(1)$ and no additional control parameters, but requires a similarity measure.

Mengsheol and Goldberg [99] have modified the deterministic crowding by suggesting a probabilistic acceptance rule. The modified algorithm is called *probabilistic crowding.* The idea is remove elitism, which may eventually lead to loss of niches due to the presence of UREs. A tournament between two individuals by the probabilistic acceptance rule gives both participants a probability to win, proportional to their fitness. Therefore, the less fit individual may be selected over the other with a higher fitness, but the probability for this to happen decreases when the difference between the fitness values increases. The authors [99] claim that this provides restorative pressure to the algorithm and prevents the loss of niches with lower fitness. However, the lack of elitism means that the algorithm may lose already found global optima.

Sharing

The concept of limiting the number of individuals of a niche not to exceed its carrying capacity was originally introduced by Holland [65, 66]. The idea is to force the individuals to share the payoff (fitness value) of each niche equally, spurring solutions away from overcrowded niches in search of new promising regions from the less populated areas. This should lead to a stable situation, where each niche contains a number of solutions proportional to its fitness. This is the foundation behind the *sharing* methods. In practice the methods modify the fitness of each individual based on its proximity to other population members.

The *fitness sharing* method of Goldberg and Richardson [56] uses the sharing concept to acquire implicit PN. The algorithm uses a sharing function to calculate a niche count between an individual and other population members.

$$sh(d_{i,j}) = \begin{cases} 1 - (\frac{d_{i,j}}{\sigma_{rad}})^{\alpha_{scale}} & \text{if } d_{i,j} < \sigma_{rad} \\ 0 & \text{otherwise} \end{cases} , \tag{3.1}$$

where $d_{i,j}$ is a distance between population members $\vec{x_i}$ and $\vec{x_j}$ defined by an appropriate distance measure, such as Euclidean distance. α_{scale} is a scaling factor to define the shape of the sharing function, often set to 1. σ_{rad} is a sharing radius, which defines the maximum distance an individual shares its fitness with others. While the presented sharing function is the most commonly used, it is possible to use other functions as well. The shared fitness f^s of an individual $\vec{x_i}$ is calculated by dividing the actual fitness f by the sum of the sharing function values between the individual and other population members:

$$f^s(\vec{x_i}) = \frac{f(\vec{x_i})}{\sum_{j=1}^{NP} sh(d_{i,j})}. \tag{3.2}$$

As the sharing function must be calculated between each generated individual and all population members, the CPT of sharing is high, $O(NP)$, equal to crowding with $CF = NP$.

A more serious drawback for sharing is the difficulty to define the sharing radius parameter [140], which is highly problem-dependent. To set the σ_{rad} properly, problem-specific information is required, which is often unavailable. For example the criterion for estimating the σ_{rad} suggested by Deb and Goldberg [36] requires information of the distances between optima, as well as their fitness values. Setting a single good value for the σ_{rad} may be very hard in problems with irregularly distributed optima. Sharing works best when the optima are approximately equally distributed in the search space, because optima within the σ_{rad} from each other become indistinguishable. Thus desired optima residing close to each other will force a small value for the σ_{rad}. Diminishing the σ_{rad} requires an increase in the population size, and the required NP may grow excessively large [55].

Mahfoud [93] demonstrates a connection between the used population size and the ability of sharing to retain optima of different fitness. An effect called genetic drift causes niches with higher fitness to draw members from other niches having lower fitness, until an equilibrium is achieved. The bigger the difference in fitness, the larger the NP required to maintain the niches with lower fitness. Sharing often requires fitness scaling [55] to emphasize the optima. Darwen and Yao [30] demonstrate that without scaling, sharing tends to create false optima around the actual optima, which prevent the algorithm from converging. On the other hand, using higher scaling power also increases the genetic drift. Too high scaling power may create super individuals, which draw the rest of the population to them too fast for sharing to work, without using excessive population sizes. This may happen even if the niches have equal fitness. Thus problem-specific information is also required to select a suitable scaling function and NP.

Cioppa et al. [27] present an iterative method to estimate optimal values for σ_{rad} and NP by using the mean and standard deviation (std) of niches found during the evolution. The method does not require a priori information of the fitness landscape. However, to achieve the optimal values, the optimization process must be run several times with varying parameter setups, which is time consuming and not practical when the goal is just to solve the optimization problem and not to analyze it further. While looking for a single good value for σ_{rad}, the method can not overcome the shortcomings of sharing in problems with closely situated optima.

Smith et al. suggest *implicit fitness sharing* [152] to overcome the difficulty of defining the σ_{rad} in a context of pattern matching. They use an immune system model, which attempts to evolve a population of antibodies to match a set of antigens. The sharing is achieved through sample-and-match procedures, which have resemblances to classifier systems. However, the method is not directly applicable to multimodal function optimization.

Menczer and Belew [98] suggest a *local selection* scheme. The scheme should not be confused with the similarly named selection scheme in section 5.3 below. The two schemes have no direct relation to each other. The local selection is not based on comparing the fitness values of individuals to each other like EAs typically work, but instead each individual competes against a fixed threshold. The population members are modeled as agents with their own energy resources. Each action an agent makes, depletes its energy resources. Agents replenish their energy resources by intaking finite resources of the environment. When an agent's energy level drops below a fixed lower threshold, it dies. On the other hand, if an agent reaches the upper energy threshold, it reproduces and shares its energy with the offspring. Competition of the resources leads to sharing-like behavior, as agents in crowded regions will run out of energy and die. Local selection is effective in maintaining the population diversity but having very low selection pressure, which prevents the algorithm from achieving convergence on some problems. The CPT of local selection is $O(1)$ for problems which allow easy maintaining of resources associated with fitness, like graph search problems. However, for continuous optimization problems the environment needs to be discretized, which increases the CPT to at least $O(NP)$.

Clearing

Another implicit PN method inspired by Holland's sharing concept is *clearing*, proposed by Pétrowski [116]. Also the *restricted competition selection* suggested by Lee et al. [81] in effect implements the clearing procedure. Clearing uses a pre-specified fixed clearing radius σ_{rad} parameter, which is similar to the sharing radius in fitness sharing. However, in clearing the σ_{rad} defines a range inside which all but the κ population members having the highest fitness are cleared. The fitness of the cleared individual is set to zero, which in the context of GA using FPS or RS prevents them from participating in the crossover and mutation operations. Contrary to sharing, which forces the population members belonging in same niche to share the resources, clearing gives all the resources to the κ best individuals of a niche.

In practice the population is first sorted in a descending order according to their fitness values. Now the first individual of the list which has the best fitness value is dominating individual of a niche. The remaining members of the list are compared to the dominating individual. The members outside the σ_{rad} range from the dominating individual do not belong to the same niche and remain unchanged. Among the members residing within the σ_{rad} from the dominating individual, the κ next remain unchanged and the rest are cleared. The process is then repeated for the rest of the population, excluding the cleared individuals and individuals which have already acted as dominating individuals until each population member has been either cleared or acted as the dominating individual.

Clearing reduces the minimum required population size compared to fitness sharing, because each niche can be maintained with only κ population members. Typically $\kappa = 1$ is used. However, defining the value for the σ_{rad} is similarly difficult as in fitness sharing. The value is problem-dependent and as sharing, clearing works best when the optima are about equally distributed in the search space. The clearing procedure is not able to maintain optima residing within the σ_{rad} range from each other. While using a small σ_{rad} does not lead to similar population explosion as in sharing, the efficiency of clearing is dependent on the value. The smaller the σ_{rad} value, the less solutions will be cleared until in an extreme case with a very small σ_{rad}, all population members will form their own niche and none will be cleared. Too small niches are inefficient in capturing the main function structure, as the AOA of a single optimum may contain several separate niches. The CPT of clearing is $O(c)$, where c is the number of niches maintained by clearing. In the extreme case when all population members form their own niche, $c = NP$ and the complexity is equal to sharing. However, typically c is considerably smaller than the population size, and thus clearing is less complex than sharing. While the CPT is not independent of the population size, it typically grows at a considerable slower rate, depending on the used σ_{rad} value.

Singh and Deb [151] propose a modified clearing approach. The idea is to reallocate the cleared solutions outside the σ_{rad} range to prevent wasting population slots and to advance the exploration of the search space. After the clearing procedure, each cleared solution which belongs to the AOA of any not cleared solution within $1.5{\cdot}\sigma_{rad}$ is randomly shifted to the region between $1.5 \cdot \sigma_{rad}$ and $3 \cdot \sigma_{rad}$ and the fitness is recalculated. After all cleared points have been considered, the clearing procedure is performed again. These additional calculations increase the CPT of modified clearing to $O(NP)$.

Im et al. [69] propose a clearing approach for ES called *restricted evolution*. They use clearing until a set of n elite solutions has been identified. Each member of the elite set is associated with its own evolution range, which corresponds to the σ_{rad} parameter. Each have similar value at the beginning, but the values change during the evolution, based on the observed improvement of fitness values inside the evolution ranges.

Clustering

Using clustering techniques for identifying niches in the population has been proposed in several studies. In implicit PN, clustering is typically used alongside with other niching techniques, like sharing or clearing, to improve their performance. For example Miller and Shaw [104] suggest a *dynamic niche sharing* method to reduce the computational complexity of sharing and increase the accuracy of identifying the niches. They use a dynamic peak identification algorithm to cluster the population to c dynamic niches, based on the condensations in population, which should correspond to peaks. Population members are classified either as a part of a dynamic niche or as non-peak individuals based on a σ_{rad} parameter, which defines the minimum distance between the peaks. Dynamic niche count

$$dnc_i = \begin{cases} nnp_j & \text{if individual i is within dynamic niche j} \\ \sum_{j=1}^{NP} sh(d(i,j)) & \text{otherwise (non-peak individual)} \end{cases}, \tag{3.3}$$

where nnp_j is the niche population size of jth dynamic niche, is used to calculate the dynamic shared fitness

$$f^d(\vec{x_i}) = \frac{f(\vec{x_i})}{dnc_i}. \tag{3.4}$$

The CPT of dynamic niche sharing starts with $O(NP)$ as most of the peaks are in the non-peak category, and reduces to $O(c)$ as the dynamic niches begin to formulate.

Various sources have suggested using clustering techniques with sharing or clearing to circumvent the difficulty of defining the sharing or clearing radius. Another advantage of such methods is that they allow varying σ_{rad}, which increases the applicability of the methods for cases with irregularly distributed optima. Yin and Germay [181] employ K-means clustering [88] prior to sharing to divide the population into niches dynamically. Sharing is only applied within the niches. The clustered shared fitness is calculated as

$$f^c(\vec{x_i}) = \frac{f(\vec{x_i})}{nnp_j \cdot (d_{i,c}/2d_{max})^{\alpha_{scale}}}, \tag{3.5}$$

where $d_{i,c}$ is the distance between individual i and the centroid of its niche, nnp_j is the number of individuals in the niche j to which individual i belongs, and d_{max} is the maximum allowed distance between an individual and its niche centroid. While the clustering method removes the need to define σ_{rad} explicitly, it is replaced by parameter d_{max}, which is not necessarily easier to specify. The CPT of the method is $O(c)$, comparable to the dynamic niche sharing method.

Gan and Warwick [46] propose *a fuzzy variable niching* technique, which maintains a separate population of overlapping fuzzy niches. Sharing is limited to individuals within a single niche. However, a single population member may belong to several niches simultaneously. Pétrowski [117] suggests the use of a clustering technique prior to the clearing. The method resembles the approach of Yin and Germay [181], the main difference being that the fitness function information is included in the clustering process to improve the ability to identify niches. The method may also be used with fitness sharing.

Clustering is sometimes also used independently of other niching methods. For example Hanagandi and Nikolaou [58] use clustering on renewing the population at regular intervals to maintain population diversity. The renewal process clusters the population. In each cluster only, the best member is saved and the rest are randomly divided into the search space.

Li et al. [84] suggest a *species conservation* technique. They cluster the population to species by using a procedure similar to clearing. However, instead of clearing the individuals residing inside σ_{rad} distance from the dominant individuals, they are marked as belonging to the same niche. The dominant individuals of each niche are called species seeds. GA is then run normally to form the population of the next generation. The species seeds are copied to the population to prevent the algorithm from losing the niches. For this, a restricted replacement is used so that the replaced individuals are the worst belonging to the same species. As the species are identified each generation, and the species seeds must be matched again in restricted replacement, the CPT of species conservation is similar to fitness sharing, $O(NP)$.

3.4.2 Explicit parallel niching

While implicit PN works with a single population, explicit PN explicitly divides the population into subpopulations and runs an optimizer for each of these independently. However, also the explicit PN methods include some degree of communication between the subpopulations. If no communication exists, explicit PN becomes equivalent to SN. Because of the interaction between the subpopulations, the boundary between implicit and explicit PN is somewhat indefinite. However, the defining feature for explicit PN methods is that they perform parallel extended local search. This means that by definition methods using explicit PN do not implement a global search phase, i.e. a search phase where the whole population information would be available for the algorithm at once. Instead, each subpopulation has only the information contained within the subpopulation at their disposal. This means that each subpopulation will concentrate its search efforts on a limited region of the whole search space, i.e. they perform an extended local search in the region defined by their subpopulation. While the lack of a global phase allows easy parallelization, it also means that the algorithms using explicit PN are not as effective as implicit PN methods in identifying and exploiting the global features of a problem.

Mating restriction

Deb and Goldberg [36] suggest a *mating restriction* concept to prevent mating between individuals from different niches, to improve the fitness sharing algorithm. The reasoning is that such pairing often produces lethals, i.e. poor quality offspring. The σ_{rad} is used in defining the range inside which mating partners are searched. Mahfoud [94] claims that the original mating restriction concept, however, is not sufficient by itself in maintaining the population diversity to offer niching capability. Explicit PN methods implement the mating restriction principle by definition. Adding a mating restriction to a method using implicit PN effectively changes it to an explicit PN method, because the global search phase is removed. Mating restriction has been suggested to be used especially with the clustering methods, which already have the population clustered as subpopulations. Yin and Germay [181] suggest a version using mating restriction to their clustering approach, using the clusters to define suitable mating partners. Similarly, Miller and Shaw [104] suggest the use of mating restriction with their dynamic niche sharing approach.

Shir and Bäck [147, 148, 149] have adopted the concept of dynamic niches for their *dynamic niching* algorithm in the context of ES. The niches are used to force the mating restriction so that each niche produces a fixed number of offspring. Shir and Bäck also propose another version of the dynamic niching based on the Covariance matrix adaptation evolution strategy (CMA-ES), which has the mating restriction as a built-in feature. Both versions require the user to define the expected number of niches c and the niche radius σ_{rad}, which defines the fixed radius of each niche and should correspond to the minimum distance between desired peaks. However, a method for adapting the σ_{rad} for each niche separately has also been suggested by Shir and Bäck [147, 150]. The idea is to make the σ_{rad} self-adaptive by coupling it to the dynamic step size of the CMA-ES algorithm. The approach also reduces the importance of the c parameter. However, it introduces two new parameters for the function of learning coefficients.

Aicholzer et al. [1] propose the use of hierarchical clustering prior to recombination in ES. Higher probability is given for both participants of a recombination to come from the same cluster than from remote clusters. The approach thus implements a probability-based mating restriction, which does not completely prevent mating between different clusters.

Stoean et al. [154] suggest an *elitist generational genetic chromodynamics* algorithm. Basically they combine the concepts of mating restriction, clearing and restricted replacement inside one algorithm. For each of these concepts, a region is defined by an appropriate radius parameter. Mating restriction is applied in the way that each individual performs crossover within the mating region. If the mating region is empty, mutation is used instead. The offspring is then pitted against the worst found solution inside a replacement region, a procedure implementing the restricted replacement concept. Finally a merging procedure, similar to clearing, is run once per generation using the merging radius. An obvious drawback of the method is the need to define three separate radius parameters. Additionally, the distance calculations mean high CPT of $O(NP)$.

Spears [153] implements the mating restriction by using *tagging*. Each population member is tagged to belong to a certain subpopulation. The subpopulations are thus not based on distance between the population members but to the genetic inheritance of an individual. Crossover is performed only between members of the same tagged subpopulation. Sharing is achieved by dividing the fitness of each individual by the number of members in its subpopulation. This removes the requirement to define the σ_{rad} parameter and the CPT of the method is $O(1)$, as no distance calculations are required. However, the number and initial distribution of labeled subpopulations must be defined instead. As the approach does not classify the subpopulation according to a distance, a potential problem arises with the small population sizes and sharing: because sharing requires a large enough population size to preserve several niches stably, dividing the population to several subpopulations decreases the effectiveness of sharing. A single optimum may draw a significant portion of individuals from several subpopulations, as sharing is not implemented between individuals belonging to different niches. This increases the already high population size requirement for sharing methods considerably. Spears notes this and suggests a modified scheme with an even tighter mating restriction as a solution, which allows mating only with the closest neighbors. The obvious risk in such an approach is that the algorithm reduces to a parallel local searcher, as the population information is only minimally used.

Island models

Island models [97] are based on the concept of punctuated equilibria, which assumes that species experience little change for most of their history and the rare evolutionary jumps happen rapidly. This is realized in algorithms using island models by separate subpopulations (islands) which evolve in isolation. After predetermined times (epochs), information is exchanged between the islands through migration. Several different variations of the island model for niching have been presented. Typically the methods require several excess parameters related to the implementation of the migration and the topology of the island structure. Lin et al. [87] categorize different approaches for the migration and island topologies. Additionally, they suggest an *injection island* GA model in which the subpopulations represent different resolution levels. Different levels in the hierarchy use different coding and the migration is one-way: from a low resolution level to a high one, which can be done without loss of information.

Bessaou et al. [8] use a speciation tree method [118] for dividing the population into subpopulations. GA is then run separately for each subpopulation, and migration is used between the subpopulations. Potter and De Jong [119] suggest *cooperative coevolution*. Species are evolved inside separate subpopulations. Interaction between the subpopulations is implemented through a domain model, which is used for calculating the fitness of an individual. Each subpopulation contributes to the domain model by sending representatives which collaborate to define the fitness. This creates an evolutionary pressure to increase the overall fitness of the population. Gustafson and Burke [57] propose a *speciating island model*, which detects solutions with a good fitness that are dissimilar to the rest of the population (outlier solutions). New subpopulations are allocated on the basis of these outlier solutions, and an EA is run concurrently for each subpopulation.

Ursem [169, 170] has adopted the island model concept for *multinational* EA. The search space (world) is divided into subpopulations (nations) which are optimized individually. A government is a subset of each nation, which consists of the nation's leading individuals. A policy is a single point calculated as a mean of the government, representing the peak the nation is approaching. A hill-valley algorithm samples points between a line drawn between two points to determine if they belong to the same peak. Two nations are merged if no valley is detected between their policies. Hill-valley detection is also used between each individual and the policy of its nation. If a valley is detected, the individual migrates to another nation whose policy is within the same peak. If no such nation is found, the individual founds a new nation. A significant downside of using the hill-valley function is the increased number of function evaluations. Each hill-valley detection requires several function evaluations. Thus each generated trial actually requires n function evaluations instead of, one as in most other population-based methods. The value of n is dependent on the accuracy used in the hill-valley detection. These extra evaluations may severely increase the complexity of the algorithm in cases where the calculation of the fitness function is time consuming, as in many real world problems. A somewhat similar idea of emerging and merging subpopulations has been presented by Streichert et al. [159]. However, instead of the hill-valley detection, they use density-based clustering [41]. This removes the need for the additional function evaluations of the multinational approach.

Niching with PSO

PSO already contains the idea of niching through the use of the best particles in the neighborhood as attractors. However, because the neighborhoods are allowed to overlap, the search will eventually converge to a single optimum [16]. To allow multimodal optimization with PSO, several different methods have been proposed. As PSO is based on independent particles which have only local interactions (except the gbest version, in which the globally best fitness is used), PN methods associated with PSO naturally belong to the category of explicit parallel niching.

Kennedy [73] and Brits et al. [16] suggest clustering with PSO. They use the cluster centers instead of the best particles in the neighborhood or the best encountered solutions located by the particle itself as attractors to encourage the formation of subswarms.

Parrot and Li [113] state that the cluster centers are not always the best performing members of the cluster, and their use as attractors may lead to poor performance. They suggest *species-based PSO* to eliminate the problem. The algorithm uses a similar method as the species conservation GA [84] for identifying the species. The species seeds are used as the neighborhood attractors for each particle belonging to the niche. The σ_{rad} parameter is required to define the niches. The CPT of the method is $O(c)$, comparable to the clearing, where c is the number of niches maintained.

Another PSO niching technique, proposed by Brit et al.[15] removes the interaction between particles by using a cognition-only model [72]. Thus each particle of the main swarm performs an independent local search. The change of the fitness of each particle is monitored, and when no improvement is noticed, a subswarm is created by combining the non-improving particle with its closest neighbor. The radius of a swarm is defined as the maximum distance between the best particle of the subswarm and any other member of the subswarm. Any main swarm member is absorbed to the subswarm if it moves inside the subswarm region. Similarly, two subswarms are merged if they intersect. A guaranteed convergence PSO [171] is run for the subswarms to achieve convergence locally.

3.5 Summary

This chapter has presented an overview of different niching methods for enabling evolutionary algorithms to perform multimodal optimization. The idea of sequential niching is to keep on restarting an optimization algorithm and to modify or constrain the search space between runs to allow the algorithms to locate different optima. The multistart methods which modify the search space between runs use sequential niching, but often GO methods are used instead of LS with SN. The main challenge of the SN methods is defining how the search space should be modified between the runs not to lose desired optima but also not to create new false optima.

Parallel niching methods have been designed to be used along with population-based optimizers. Explicit PN methods divide the search space into subpopulations and run the optimizer for each subpopulation independently. This allows easy parallelization, but forbids the algorithms from identifying and exploiting global information efficiently. Implicit PN methods change the optimization algorithm, encouraging the preservation of population diversity. As the whole search space is considered at once, implicit PN methods potentially allow identification and exploitation of global information.

In this thesis, the complexity of the niching methods is estimated using complexity per trial, which offers a more intuitive complexity measure compared to the often used generational complexity. The next chapter concentrates on issues related to evaluating optimization algorithms experimentally especially in the context of continuous multimodal optimization.

Chapter IV

Evaluating optimization algorithms

Achieving a reliable comparison between two global optimization algorithms is not an easy task. The first challenge is the definition of end condition for the search. A typical end condition is the *maximum allowed number of function evaluations*, NFE_{max}, which is defined for each tested problem. Using too low NFE_{max} compared to the difficulty of the function favors greedy methods: a greedy algorithm may occasionally find the global optimum very fast, while a less greedy approach fails to achieve convergence within the given time frame. Increasing the value of NFE_{max} may change the situation so that the greedy approach still fails a majority of the runs due to premature convergence, but the less greedy approach has now enough time to converge and is always able to locate the global optimum. The downside of increasing NFE_{max} is that it will increase the running time of the tests. Thus a rather low NFE_{max} has to be often chosen due to practical reasons, which limits how difficult test functions can be used. Also the used performance measures must be considered carefully, so that the achieved results display all the important characteristics of the compared algorithms.

Another difficulty in comparing the algorithms is the definition of control parameters. Each method has typically a specific set of control parameters, which have a great effect on the performance of the algorithm. Good values for these parameters are often problem-dependent. Running the same algorithm with two different control parameter setups may produce a greater difference in performance than comparing two different algorithms with certain parameter setups. For this reason, a comprehensive parameter study is an essential part of evaluating algorithms experimentally. Simply comparing algorithms using only one or few parameter setups is likely to reveal little relevant information of the properties of the algorithms and can be misleading, if one method happens to have a clearly more suitable parameter setup for the problem at hand. Searching at least coarsely a good parameter setup for each problem algorithm pair reveals the true potential of the algorithm in connection with such problems. The parameter study also reveals the sensitiveness of an algorithm to its control parameters. Unfortunately, such studies are time consuming and thus often neglected.

Most importantly, the features of the test problems should be known to make a useful conclusion on the performance of the algorithm. The no free lunch theorems [179, 180] state that no optimization algorithm can outperform another over the set of all possible problems, unless one of the algorithms is able to reduce the number of redundant points searched. So, to be able to claim that an algorithm outperforms another, also the subset of problems for which this holds must be defined. In order to compare algorithms, their ability to identify and exploit different problem features must be compared. For instance, Whitley [175] has demonstrated the ability of different optimization algorithms to exploit different problem features.

4.1 Important problem features

This section details the most important problem features in the context of this thesis, emphasizing especially features related to multimodal optimization. Features and issues related to multiobjective optimization or the optimization of problems containing constraint functions other than simple box constraints are outside the scope of this thesis and not considered. Additionally, the listed features present coding-independent characteristics for continuous optimization and coding-related issues (see for example [174]) are not considered.

4.1.1 Dimensionality and the number of optima

When the dimensionality of the search space increases, its size grows exponentially. While it is possible to cover the search space extensively in low dimensional functions, this becomes increasingly difficult as the dimensionality increases. Typically for the multimodal test functions used in the literature, the number of optima increases along with the number of dimensions, although exceptions exist, like the Griewangk function [176]. The number of optima is one of the most important features of a function, especially in the context of multimodal optimization, as it affects also the goal of the optimization. Low dimensionality and a low number of optima favor methods that rely heavily on extensive coverage of the search space.

4.1.2 Relative area of attraction sizes

The size of the area of attraction of an optimum directly indicates the probability for a sample to be placed there by random placement. Once a point is successfully placed inside the AOA of each interesting optimum, running a local optimization algorithm from each of the sample points produces the solution. When the differences in the relative AOA between the optima increase, the optima with smaller AOAs will become harder to find.

4.1.3 Relative fitness of optima

The differences between the fitnesses of the optima are important for niching methods, especially when the goal is to locate local optima as well. Typically the chance of a niching method to maintain a local optimum is proportional to the fitness of the optimum. For example, Mahfoud [93] explains that for methods using fitness sharing, the minimum population size required to maintain a local optimum of certain fitness is proportional to the relative fitness as well as the number of optima with better fitness.

4.1.4 Separability

Separability is a synonym for decomposability. In separable functions the parameters are independent of each other, i.e. “there are no nonlinear interactions between variables” [176, p. 246]. A separable function can be optimized by optimizing each parameter individually. A nonseparable function is naturally the opposite of a separable function, meaning that parameters are not independent, and optimizing parameters individually no longer works. A separable function can be made nonseparable by rotation. Salomon [139] demonstrates that the complexity of finding optima of separable functions increases linearly along with the increase in dimensionality. For nonseparable functions, the increase is typically exponential.

Epistasis [6] is a measure of separability. It measures the number of nonlinear interactions between the variables. A separable function has minimal epistasis, and the amount of epistasis increases with the number of nonlinear interactions. Typically evolutionary algorithms using crossover operators are capable of exploiting separability (low epistasis).

4.1.5 Directional bias and regularity

Directional bias [92] in multimodal problems means that the optima are located in direct lines. If these lines correspond to the axis directions, the problem becomes separable. While rotation can be used to remove the separability, the directional bias will remain. While not as readily exploitable as separability, an algorithm that is able to adapt itself to the rotation can still benefit from the directional bias.

A function is regular if in addition to having a directional bias, all the optima are of equal distance from each other. Thus for each dimension, only a single distance between the optimum points exists. Partially regular functions have several different distances along a single dimension between optima, at least in some of the dimensions. In irregular functions, all the distances between each two optima are unique. The degree of regularity represents the proportion of identical distances along each dimension from the total number of such distances. The degree of regularity is maximal in regular functions which only contain a single distance for each dimension and minimal in irregular functions where the number of identical distances is zero.

4.1.6 Symmetry

A two-dimensional function is symmetric if $F(x_1, x_2) = F(x_2, x_1)$ [176]. Such equivalences may exist for a D dimensional function up to $D!$. Symmetry can be exploited by exchanging the variables of a solution to locate symmetrically positioned solutions. Rotationally symmetric functions remain symmetric regardless of rotation. For example Schaffer’s F6 function [92, 163] is rotationally symmetric.

4.1.7 Continuity and differentiability

A continuous function does not contain points of discontinuity. Classical optimization methods often require the function to be differentiable, meaning that a meaningful value for a derivative can be calculated for any point. Thus in addition to being continuous, the function must not have discontinuities in the slope.

4.1.8 Global structure

An important characteristic of a function is the overall global structure. For example, the Rosenbrock function (eq. 4.8) is characterized by a banana shaped valley which leads to the optima at $[1, 1, \ldots, 1]$. Many traditional test functions, like Rastrigin (eq. 4.9) or Griewangk, have the global optimum at origin in the middle of a parabolic valley in the center of the search space. This allows a global optimization algorithm to estimate the direction to the global optimum by exploiting the global structure of the function. Additionally, methods based on averaging have typically a search bias towards the center, which allows the algorithm to exploit the centered global optimum. Another way of exploiting the location of an optimum is by copying parameter values from one dimension to another in cases where all the parameters are of equal value in the global optimum, as in the example functions above.

4.1.9 Optima on constraints

Optima located on constraints may pose a problem for some optimization methods, like the grid-based gradient descent (GRGD) implementation used in this thesis. On the other hand, the location of optima on constraints is exploitable because the points on the constraints represent only a part of the whole search space. Any algorithm concentrating its efforts on the constraints will thus have an advantage locating such optima.

4.2 Performance measures

The performance of a global optimization algorithm can be experimentally evaluated using several different performance measures. Selecting a suitable performance measure is also important for high quality experimental studies. Poorly chosen performance measures may concentrate in features of secondary importance and produce misleading results, similarly as a poorly chosen test setup. The term *effectiveness* is typically associated with solution quality, while *efficiency* refers to the speed of the algorithm. Typically for stochastic methods, the results are reported as averages from a certain number of independent runs. Statistical tests should be used in conjunction to demonstrate the significance of the results.

4.2.1 Locating a single optimum

Usually either the progress of an algorithm as a function of time, or the quality of a solution obtained within a fixed time frame is measured. While the execution time can be used directly, it is more common to use the performed *number of function evaluations*

(NFE) as the measure of time. The problem of using actual time is that the measure is then dependent on the implementation as well as the system where the experiments were performed, and the results become difficult to reproduce and compare to by other researchers. NFE, on the other hand, does not take into account the complexity of the used optimizers and thus results using actual time can be used to complement results attained with a measure using NFE. Using solely the actual time can be misleading, however, because typically functions used to test the performance of an optimizer have low evaluation cost, and the evaluation time plays often a minor role compared to the complexity of the algorithm itself. In real-world problems the situation often changes, and the cost of function evaluations becomes the dominating factor. In such situations an algorithm which requires a low number of function evaluations but has otherwise high computational complexity is preferable to one which has low complexity, but requires more function evaluations.

A simple way of demonstrating the performance is to plot the progress of an algorithm as a function of time, or record the best achieved value after a termination condition is fulfilled. This is especially useful when using problems for which the actual global optimum is not known beforehand. Often, in addition to reporting the average performance, also the worst and best performing runs are documented.

Success rate (SR) defines the fraction of the test runs which are able to locate the global optimum with precision ε within the given time frame

$$SR = \frac{number\ of\ successful\ runs}{total\ number\ of\ test\ runs}. \tag{4.1}$$

The measure is often used in test problems with a known global optimum. ε states the maximum allowed difference in fitness value compared to the known globally optimal value, but the location can also be defined by coordinates. Additionally, the NFE to achieve the globally optimal solution is typically recorded to measure the speed of an algorithm.

Success performance (SP) was used in the CEC 2005 contest for real parameter optimization. It aims to combine both the speed and reliability of an algorithm into a single performance measure. It is calculated by dividing the average NFE by SR

$$SP = \frac{average\ NFE}{SR}. \tag{4.2}$$

SP is independent of NFE_{max} with assumption that large enough value is used not to halt potentially successful runs. The measure considers two methods to be equally effective if, for example, one method finds the solution half as often but twice as fast as the other.

Another important, but qualitative performance measure is the difficulty of defining good values for the control parameters of an algorithm. High number of control parameters does not necessarily mean that an algorithm is difficult to tune for different problems. More important is the existence of problem dependent parameters, for which general recommendations can not be given without problem-specific information. Additionally the sensitiveness of the algorithm for slight changes in parameter values should be measured. Highly sensitive parameters which in addition are problem dependent are very hard to tune correctly. An example of such an difficult parameter is the σ_{rad} used by many niching methods.

4.2.2 Multimodal optimization

Before being able to measure the performance of a multimodal optimization algorithm, the exact goal must be defined: which of the optima is the algorithm supposed to find? Typically all global optima are of interest. In test problems with only a few optima, the goal is often defined to locate them all. In more complex problems, the locally global optima are typically of interest, in addition to the global ones. Once the desired set of optima is defined, the performance of an algorithm on locating them can be measured.

Success rate is often used also in multimodal optimization, where it defines the fraction of test runs which were able to locate the full set of desired optima. *Peak ratio* (PR) [163]

$$PR = \frac{number\ of\ optima\ found}{total\ number\ of\ optima} \tag{4.3}$$

defines the fraction of found optima from the total number of optima in the desired set. PR is often drawn as a function of time to see how the number of found optima develops and whether the algorithm will start to lose already found optima eventually. The NFE to locate a certain number of optima is typically also recorded to measure speed. Drawing the NFE to locate the i:th optimum as a function of the number of desired optima is a good way to evaluate the progress of a multimodal optimization algorithm. For an example, see Figure 6.2.

In simple one or two-dimensional functions, the population can be drawn on top of the function graph to demonstrate the distribution. Several such snapshots taken during the search can be used to demonstrate the progress of population. Deb and Goldberg [36] have introduced a *chi-square like* performance measure for comparing the population distribution to an ideal proportionally populated distribution. The measure indicates the ability of an algorithm to populate niches proportionally to their fitness, which is important for niching methods based on the sharing principle.

4.3 Experimental study of DE and G3-PCX

The difficulties in comparing optimizers, as well as the importance of selecting a test set properly to exhibit the desired features for comparing algorithms are demonstrated by using the results originally published by the author in [133] as an example. Deb et al. [35, 34] propose a real-parameter genetic algorithm called the generalized generation gap (G3) using the generic parent-centric recombination (PCX) operator. The PCX operator is shown to outperform previous unimodal normal distribution crossover and simplex crossover operators. Most interestingly, the G3-PCX algorithm is compared to several optimization algorithms, including Differential Evolution. The reported results suggest the G3-PCX to be able to outperform DE by an order of magnitude in a test setup consisting of three differentiable test functions, two of which were unimodal and the last consisting up to two optima. Typically such problems can be efficiently solved by using local optimization algorithms, as good performance in unimodal functions often requires a particularly greedy algorithm. Typically, however, greedy methods have a tendency for premature convergence in multimodal problems. While the multimodal Rastrigin function (eq. 4.9) was briefly studied with the G3-PCX, no comparisons to the other algorithms were reported. This experimental study adds to the investigation of Deb et al. [35, 34] by extending the comparison of G3-PCX and DE on highly multimodal problems, and demonstrates that the features of selected test problems play a crucial role in determining the outcome of such comparisons between different algorithms.

4.3.1 The algorithms

Differential Evolution [125] is a population-based evolutionary algorithm designed for global optimization. The particular strategy used in this comparison is the DE/rand/1/bin. The algorithm uses vector differentials scaled by parameter F to determine a mutation step length. This makes the algorithm exhibit self-adaptive behavior, as the average mutation step length decreases as the population converges. The algorithm also uses a uniform crossover, whose frequency is controlled by parameter CR so that $CR = 1$ disables the crossover. Small values for CR allow the algorithm to exploit low epistasis of a function efficiently. DE is described in detail in Chapter 5 of this thesis.

The G3-PCX algorithm is a steady state EA described as follows [35]:

1. From a population of individuals, select the best individual and $\mu - 1$ other individuals randomly to be parents.
2. Generate λ offspring from the chosen μ parents using the PCX recombination scheme.
3. Select one individual $\vec{x}_i$ at random from the population and combine this with λ offspring to make a subpopulation.
4. Select the best solution of the subpopulation and replace the selected individual $\vec{x}_i$ with this.

In the PCX recombination scheme the offspring $\vec{u}$ is calculated as

$$\vec{u} = \vec{x}_p + \omega_\zeta \vec{d}_p + \sum_{i=1, i \neq p}^{\mu} \omega_\eta \bar{D} \vec{e}_i, \tag{4.4}$$

where $\vec{x}_p$ is the best individual from the current population, as also in [35], to achieve fast convergence, $\vec{d}_p$ is a direction vector from the mean of μ parents to $\vec{x}_p$, $\bar{D}$ is an average of perpendicular distances to the line defined by $\vec{d}_p$ from $\mu - 1$ other parents than $\vec{x}_p$, and $\vec{e}_i$ are $\mu - 1$ orthonormal bases that span the subspace perpendicular to $\vec{d}_p$. Coefficients ω_ζ and ω_η are zero-mean normally distributed random variables with variances σ_ζ^2 and σ_η^2.

PCX creates an offspring that has a greater probability of being closer to parent $\vec{x}_p$ than far from it. Parameters σ_ζ^2 and σ_η^2 control how much the offspring vary from $\vec{x}_p$ in the direction of $\vec{d}_p$ and the direction perpendicular to $\vec{d}_p$. The dependence of $\bar{D}$ and length of $\vec{d}_p$ on mutual distances between selected μ parents, makes PCX self-adaptive similarly as DE.

4.3.2 Function setup

A set of 12 test functions were used in comparing the algorithms including four unimodal, three separable multimodal and five nonseparable multimodal functions.

Unimodal functions

The sphere function [33]

$$f_{sph}(\vec{x}) = \sum_{i=1}^{D} {x_i}^2 \tag{4.5}$$

is the simplest quadratic function to minimize. The function is unimodal and separable.

The ellipse function [35]

$$f_{ell}(\vec{x}) = \sum_{i=1}^{D} i \cdot {x_i}^2 \tag{4.6}$$

tests the ability of a method to optimize a function whose parameters have disparate magnitudes. The function is unimodal and separable.

The rotated ellipsoid function (f_{sch} in [35]), also known as Schwefel's ridge

$$f_{rel}(\vec{x}) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{4.7}$$

rotates the ellipse and tests the ability of an algorithm to optimize an unimodal, non-separable function.

The generalized Rosenbrock function [35]

$$f_{ros}(\vec{x}) = \sum_{i=1}^{D-1} \left(100({x_i}^2 - x_{i+1})^2 + (1 - x_i)^2\right) \tag{4.8}$$

is a fourth order polynomial whose basin is non-convex and often referred to as being unimodal. Shang and Qiu [145] have proved that there exists one local minimum in addition to the global minimum with $4 \leq D \leq 30$ and leaves open the possibility of additional local minima with higher dimensions. The function is characterized by a banana-shaped valley which leads to the global optimum.

Separable multimodal functions

The generalized Rastrigin function [168]

$$f_{ras}(\vec{x}) = 10D + \sum_{i=1}^{D} \left({x_i}^2 - 10\cos(2\pi x_i)\right) \tag{4.9}$$

is a highly multimodal regular function with a global minimum in the origin.

The normalized Schwefel function [143]

$$f_{sch}(\vec{x}) = \frac{\sum_{i=1}^{D} -x_i \sin(\sqrt{|x_i|})}{D} \tag{4.10}$$

is a constrained ($-512 \leq x_i \leq 512, \quad i = 1, \ldots, D$) regular multimodal problem with a global minimum close to the constraints.

The Ackley function [185]

$$f_{ack}(\vec{x}) = \sum_{i=1}^{D-1} \left(e^{-0.2} \sqrt{{x_i}^2 + {x_{i+1}}^2} + 3\left(\cos(2x_i) + \sin(2x_{i+1})\right) \right) \tag{4.11}$$

is a highly multimodal regular function with two global minima close to the origin.

Nonseparable multimodal functions

The Salomon function [125]

$$f_{sal}(\vec{x}) = -\cos\left(2\pi \sqrt{\sum_{i=1}^{D} x_i^2}\right) + 0.1 \sqrt{\sum_{i=1}^{D} x_i^2} + 1 \tag{4.12}$$

is a rotationally symmetric function with a circular cosine wave around the global minimum in the center and the local optima forming circles around the global optimum in the middle.

The Whitley function [125]

$$f_{whi}(\vec{x}) = \sum_{i=1}^{D}\sum_{j=1}^{D} \left(\frac{\left(100(x_i^2 - x_j)^2 + (1 - x_j)^2\right)^2}{4000} - \cos\left(100(x_i^2 - x_j)^2 + (1 - x_j)^2\right) + 1 \right) \tag{4.13}$$

combines a very steep overall slope with a highly multimodal area around the global minimum located at $x_i = 1, \quad i = 1, \ldots, D$.

The modified Langerman function [125]

$$f_{lan}(\vec{x}) = -\sum_{i=1}^{5} \gamma_i e^{-\frac{1}{\pi}\sum_{j=1}^{D}(x_j - a_{ij})^2} \cos\left(\pi \sum_{j=1}^{D} (x_j - a_{ij})^2\right), \tag{4.14}$$

where

$$\mathbf{A} = \begin{pmatrix} 9.681 & 0.667 & 4.783 & 9.095 & 3.517 & 9.325 & 6.544 & 0.211 & 5.122 & 2.020 \\ 9.400 & 2.041 & 3.788 & 7.931 & 2.882 & 2.672 & 3.568 & 1.284 & 7.033 & 7.374 \\ 8.025 & 9.152 & 5.114 & 7.621 & 4.564 & 4.711 & 2.996 & 6.126 & 0.734 & 4.982 \\ 2.196 & 0.415 & 5.649 & 6.979 & 9.510 & 9.166 & 6.304 & 6.054 & 9.377 & 1.426 \\ 8.074 & 8.777 & 3.467 & 1.863 & 6.708 & 6.349 & 4.534 & 0.276 & 7.633 & 1.567 \end{pmatrix}$$

and

$$\vec{\gamma} = \begin{pmatrix} 0.806 & 0.517 & 0.100 & 0.908 & 0.965 \end{pmatrix},$$

has three condensations of minima in an otherwise flat surface.

The modified Shekel Foxholes function [125]

$$f_{she}(\vec{x}) = -\sum_{i=1}^{30} \frac{1}{\sum_{j=1}^{D}(x_j - a_{ij})^2 + \gamma_i}, \tag{4.15}$$

where

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

and

$$\vec{\gamma} = \Big(\begin{array}{ccccccccccc}
0.806 & 0.517 & 0.100 & 0.908 & 0.965 & 0.669 & 0.524 & 0.902 & 0.531 & 0.876 & 0.462 \\
0.491 & 0.463 & 0.714 & 0.352 & 0.869 & 0.813 & 0.811 & 0.828 & 0.964 & 0.789 & 0.360 \\
0.369 & 0.992 & 0.332 & 0.817 & 0.632 & 0.883 & 0.608 & 0.326 & \Big), & &
\end{array}$$

has a narrow minimum peak in a corner of optima condensation in an otherwise flat surface.

The normalized Rana function extended with a diagonal wrap [125]

$$f_{rana}(\vec{x}) = \left(\sum_{i=1}^{D} \Big(x_i \sin\left(\sqrt{|x_j + 1 - x_i|}\right) \cos\left(\sqrt{|x_j + 1 + x_i|}\right) + \right.$$
$$\left. (x_j + 1) \cos\left(\sqrt{|x_j + 1 - x_i|}\right) \sin\left(\sqrt{|x_j + 1 + x_i|}\right) \Big) \right) / D \tag{4.16}$$

where $j = (i+1) \mod D$ is a particularly difficult constrained ($-520 \leq x_i \leq 520, \quad i = 1, \ldots, D$) problem with the global minimum close to the border. The exact function descriptions and figures of two-dimensional versions are available in `http://www.it.lut.fi/ip/evo/functions/`.

4.3.3 Test setup

A set of test runs was conducted with different control parameter settings for both algorithms. With DE, all combinations of 0.2, 0.5, and 0.9 were used for *F* and *CR* to compare how the algorithm performs with small, medium, and large values for both parameters in different problems, while *NP* was varied.

For the G3-PCX algorithm, the first test set was done by fixing $\sigma_\zeta = 0.1$ and $\sigma_\eta = 0.1$, as suggested by the authors [35]. For λ, values of 2, 4, and 10 and for μ, values of 3, 6, and 10 were used. Because the algorithm could not solve some of the problems with any of these combinations, a second test set was conducted, fixing $\lambda = 2$ and $\mu = 3$, and varying the values for σ_ζ and σ_η from 0.1 to 1.

With each *NP*, 10-1000 independent runs were conducted according to the problem. The convergence speed and percentage of failed executions of the DE algorithm were compared to the G3-PCX model. Two-dimensional versions were used for all problems. The effect of increasing the dimensionality to five was studied with all, but f_{lan}, f_{she} and f_{rana}.

SR was calculated using $\varepsilon = 10^{-5}$. In some cases also $\varepsilon = 10^{-20}$ was used to make the results comparable to the ones presented in [35]. The value for NFE_{max} (Table 4.1) was experimentally selected to be considerably larger compared to the typical number of function evaluations required to solve the problem. Such a large value was selected to acquire a good estimate of the percentage of runs that really converge prematurely rather than favoring the settings that lead to fast convergence. The algorithms were prevented from expanding their search outside the constraints used for initializing the population by using boundary constraints in all problems, except for tests with skewed initialization with the Rastrigin function. The constraints were handled simply by replacing the faulty variable with a randomly selected value inside the constrained area.

Table 4.1: Settings for different problems.

Function	Dimension	NFE_{max}	Used constraints
$f_{sph,ell,rel,ros}$	2 and 5	$5 * 10^5$	$-5.12 \leq x_j \leq 5.12 \quad j = 1, \ldots, D$
$f_{ras,whi}$	2	$5 * 10^5$	$-5.12 \leq x_j \leq 5.12 \quad j = 1, \ldots, D$
$f_{ras,whi}$	5	$2 * 10^6$	$-5.12 \leq x_j \leq 5.12 \quad j = 1, \ldots, D$
f_{sch}	2	$5 * 10^5$	$-512 \leq x_j \leq 512 \quad j = 1, \ldots, D$
f_{sch}	5	$2 * 10^6$	$-512 \leq x_j \leq 512 \quad j = 1, \ldots, D$
$f_{ack,sal}$	2	$5 * 10^5$	$-30 \leq x_j \leq 30 \quad j = 1, \ldots, D$
f_{ack}	5	$2 * 10^6$	$-30 \leq x_j \leq 30 \quad j = 1, \ldots, D$
f_{sal}	5	10^7	$-30 \leq x_j \leq 30 \quad j = 1, \ldots, D$
$f_{lan,she}$	2	$5 * 10^5$	$-5 \leq x_j \leq 15 \quad j = 1, \ldots, D$
f_{rana}	2	$2 * 10^6$	$-520 \leq x_j \leq 520 \quad j = 1, \ldots, D$

4.3.4 Results

The full set of results is available in `http://www.it.lut.fi/ip/evo/`, including standard deviations. The deviations are typically rather small, suggesting the results to be considerably stable.

Unimodal Problems

Very similar behavior was observed in all convex unimodal functions. Both algorithms were able to find the global optimum without error with reasonable population sizes, regardless of the control parameter setup. An example of typical performance in unimodal problems is shown in Figure 4.1. G3-PCX was clearly faster when the slope of the curve representing the convergence speed in the function of *NP* (NFE_{slope}) is considered. Increasing the dimensionality of problems from two to five or using $\varepsilon = 10^{-20}$ instead of $\varepsilon = 10^{-5}$ decreased the convergence speed. The relative performance difference between the algorithms and the best performing parameter settings did not change significantly.

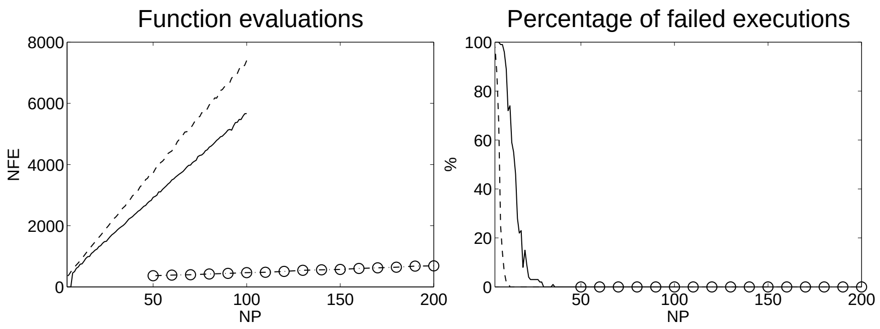


Figure 4.1: Average number of function evaluations required to reach the global optimum, and the percentage of failed runs for f_{ell} with $D = 5$ and $\varepsilon = 10^{-5}$. DE, $F = 0.2, CR = 0.2$ ('−'); DE, $F = 0.5, CR = 0.2$ ('−−'); G3-PCX, $\lambda = 2, \mu = 3, \sigma_\zeta = 0.1, \sigma_\eta = 0.1$ ('$\cdot - \circ - \cdot$').

For the f_{ros}, the results of the two-dimensional version were in line with the results acquired with the convex unimodal problems. The situation changed when the dimensionality was increased to five (Figure 4.2), and the function gained the second minimum. While DE was again able to reach zero in error with most control parameter settings, G3-PCX was not able to solve the problem without failed runs with any of the tested control parameter settings. The increase in *NP* did not clearly decrease the percentage of failures. DE had serious problems with solving the function with a small value for F. For runs with small *CR*, $NFE_{max} = 5 * 10^5$ was not large enough with large population sizes. For successful runs, NFE_{slope} for G3-PCX was still clearly gentler compared to DE.

G3-PCX achieved the best performance with the control parameter setup $\lambda = 2, \mu = 3, \sigma_\zeta = 0.1, \sigma_\eta = 0.1$ in all unimodal cases. However, using larger values for σ_ζ and σ_η slightly decreased the failed runs with the five-dimensional Rosenbrock, but the price was clearly decreased convergence speed. Almost identical behavior in f_{ell} and f_{rel} confirmed that G3-PCX is rotationally invariant and does not exploit the possible separability of a function.

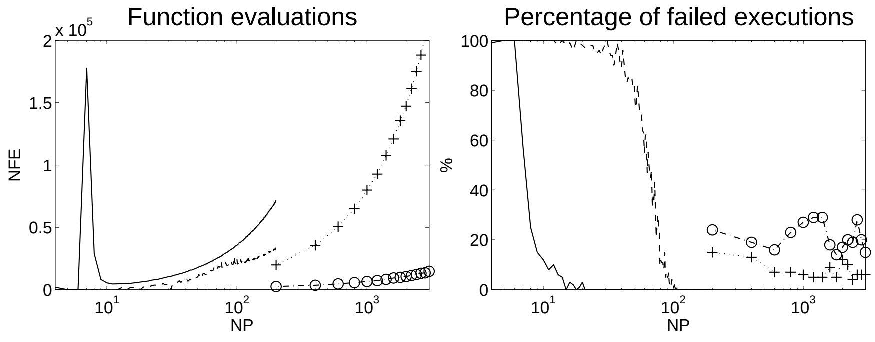


Figure 4.2: Average number of function evaluations required to reach the global optimum, and the percentage of failed runs for f_{ros} with $D = 5$ and $\varepsilon = 10^{-5}$. DE, $F = 0.9, CR = 0.9$ ('$-$'); DE, $F = 0.5, CR = 0.9$ ('$--$'); G3-PCX, $\lambda = 2, \mu = 3, \sigma_\zeta = 0.1, \sigma_\eta = 0.1$ ('$\cdot - \circ - \cdot$'); G3-PCX, $\lambda = 2, \mu = 3, \sigma_\zeta = 1, \sigma_\eta = 1$ ('$\cdot \cdot + \cdot \cdot$').

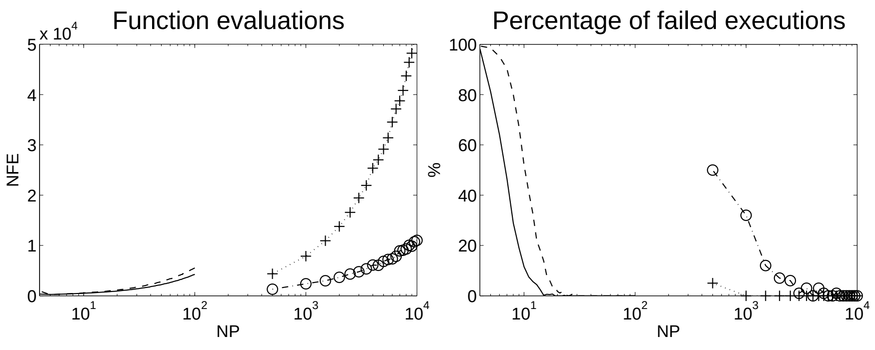
For DE, small *CR* worked well with the separable f_{sph} and f_{ell} functions, while the nonseparable f_{rel} and f_{ros} benefited from large *CR*. This was expected, as small *CR* allows DE to exploit the low epistasis of separable functions. A larger F enabled the algorithm to find the optimum reliable with a smaller population size, but the NFE_{slope} was then higher. This means that with overly large *NP*, the algorithm works faster with smaller F. Interestingly, using small *CR* seemed to allow the use of smaller *NP* in both separable and nonseparable functions. The effect was especially visible with a small F.

Separable Multimodal Problems

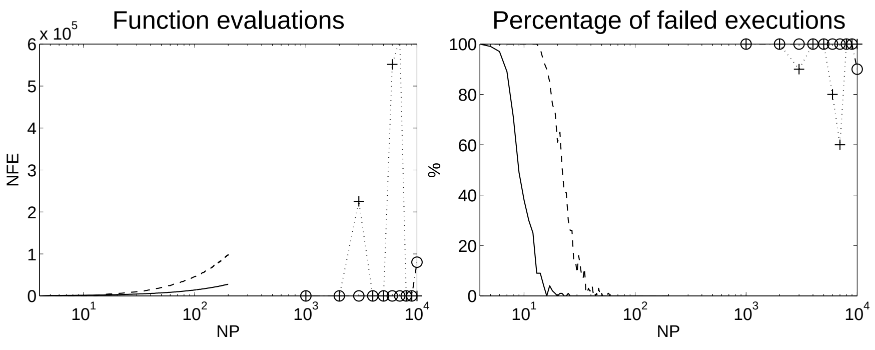
DE was able to solve all separable multimodal problems without error with all the tested control parameter setups, provided that *NP* was large enough. G3-PCX, however, had serious problems with all the problems. In two-dimensional cases, G3-PCX was able to achieve zero *failure rate* (failure rate means the percentage of failures which stays constant as *NP* increases) with most parameter setups, but rather large population sizes were needed. G3-PCX was still able to achieve gentler NFE_{slope}, but the increased *NP* requirement made the convergence speeds rather comparable (Figure 4.3(a)).

When the dimensionality was increased to five, G3-PCX was no longer capable to solve the problems reliably. A maximum population of 10^4 was not enough for G3-PCX to come even close to zero failure rate in f_{ras} and f_{sch}. Especially in function f_{ras} G3-PCX had difficulty in finding the optimum at all, as can be seen in figure 4.3(b). In f_{ack}, G3-PCX demonstrated somewhat better results (Figure 4.4), probably because of the two global optima which increased the chances of locating either one, compared to Rastrigin, which is otherwise rather similar.

The increase of values σ_ζ and σ_η clearly increased the percentage of successful runs, but still not enough to reach zero failure rates, and the convergence speed decreased significantly. The NFE_{slope} for DE was steeper but G3-PCX required a very large *NP*



(a) $D = 2$



(b) $D = 5$

Figure 4.3: Average number of function evaluations required to reach the global optimum, and the percentage of failed runs for f_{ras} with $\varepsilon = 10^{-5}$. DE, $F = 0.5, CR = 0.2$ ('−'); DE, $F = 0.5, CR = 0.9$ ('−−'); G3-PCX, $\lambda = 2, \mu = 3, \sigma_\zeta = 0.1, \sigma_\eta = 0.5$ ('$\cdot - \circ - \cdot$'); G3-PCX, $\lambda = 2, \mu = 3, \sigma_\zeta = 1, \sigma_\eta = 1$('$\cdot \cdot + \cdot \cdot$').

to reach even a reasonable percentage of successful runs. DE thus found the optimum with fewer function evaluations compared to G3-PCX. Parameters λ and μ appeared to have only a negligible effect on the failure rate in this type of problems, but larger values decreased the convergence speed significantly.

As expected, small *CR* again worked best with DE. The advantage compared to using large *CR* increased with dimensionality, as can be seen in Figure 4.3. Parameter F behaved similarly as in unimodal functions: a smaller value produced gentler NFE_{slope} but required larger *NP* to solve the problem reliably. The connection between smaller *CR* and a smaller minimum usable population size was seen also with the separable multimodal problems. The combination of large *CR* and small F required larger *NP* to find the optimum reliably compared to other control parameter setups.

A limited test setup with DE on f_{ras} without constraints and using skewed initialization $x_i \in [-10, -5]$ not including a global optimum, was conducted as in [35]. Dimensions

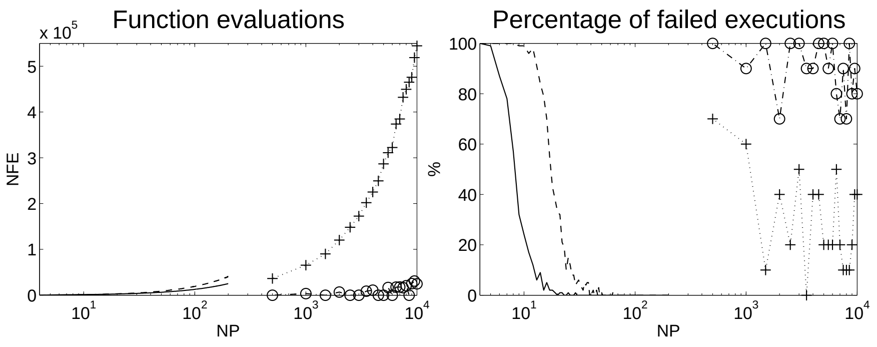


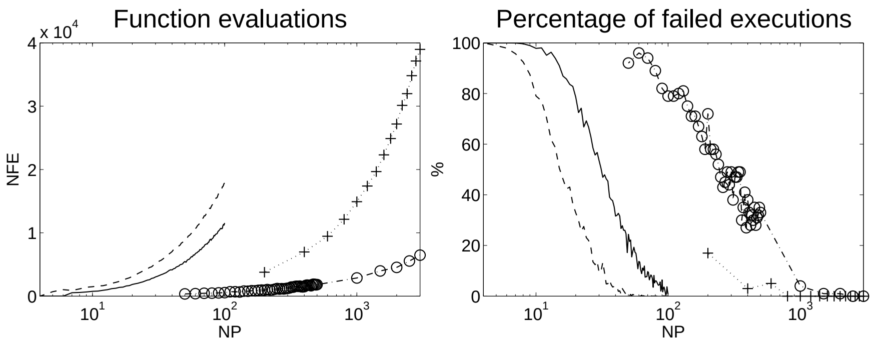
Figure 4.4: Average number of function evaluations required to reach the global optimum, and the percentage of failed runs for f_{ack} with $D = 5$ and $\varepsilon = 10^{-5}$. DE, $F = 0.5, CR = 0.2$ ('−'); DE, $F = 0.5, CR = 0.9$ ('−−'); G3-PCX, $\lambda = 2, \mu = 3, \sigma_\zeta = 0.1, \sigma_\eta = 0.1$ ('$\cdot - \circ - \cdot$'); G3-PCX, $\lambda = 2, \mu = 3, \sigma_\zeta = 1, \sigma_\eta = 1$ ('$\cdot \cdot + \cdot \cdot$').

2, 5, and 20 were tested. The number of required function evaluations increased in two- and five-dimensional cases compared to a similar control parameter setup with normal initialization, but the relative differences between different control parameter setups remained similar. The failure rates did not change much with large F, but when F decreased, the requirement for a minimum population size increased and small values kept DE from solving the problem at all. This is logical, because with a too small mutation step length, the population is unable to move to the new area where the global optimum is. DE was also able to solve the 20-dimensional version rather easily. The best results were acquired using small CR and large F. Using parameter setup $CR = 0.2$, $F = 0.9$, and $NP = 100$, for example, global optimum with $\varepsilon = 10^{-5}$ was found each time in 100 independent runs with average NFE of 166040. For comparison, G3-PCX was reported to fail to solve the 20-dimensional Rastrigin using a similar setup in [35]. However, also DE had difficulties with large CR as it could not exploit the separability efficiently.

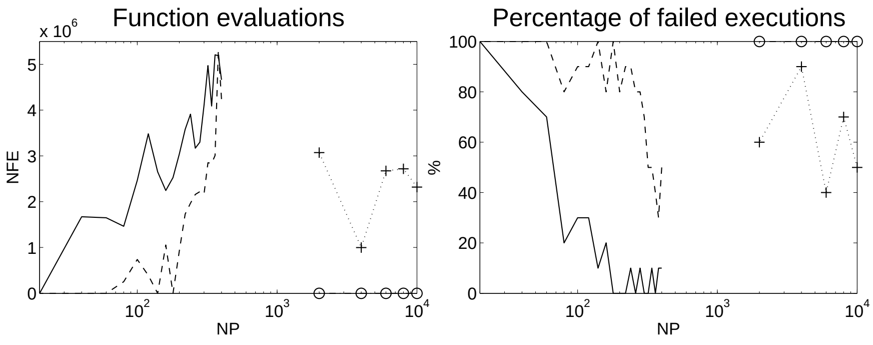
Nonseparable multimodal problems

DE attained the zero failure rate with proper parameter settings in most nonseparable multimodal problems. The five-dimensional Salomon function, however, caused the algorithm some difficulties. As seen in Figure 4.5(b), the optimization process requires a large number of function evaluations to reach the global optimum. Values of $NFE_{max} = 10^7$ and the maximum population size of 400 were apparently not enough for the problem with all control parameter setups. A rather small failure rate (below 10%) was achieved using parameters $F = 0.2$ and $CR = 0.5$. Another difficult function was Rana (Figure 4.6). The failure rate, however, approached zero steadily with all control parameter settings, and using a larger NP would be likely to result in reaching the zero.

G3-PCX had again difficulties in achieving low failure rates. In functions f_{sal}, f_{whi}, and f_{she} with $D = 2$, the algorithm reached the zero failure rate and acquired performance



(a) $D = 2$



(b) $D = 5$

Figure 4.5: Average number of function evaluations required to reach the global optimum, and the percentage of failed runs for f_{sal} with $\varepsilon = 10^{-5}$. DE, $F = 0.2, CR = 0.5$ ('−'); DE, $F = 0.9, CR = 0.9$ ('−−'); G3-PCX, $\lambda = 2, \mu = 3, \sigma_\zeta = 0.1, \sigma_\eta = 0.1$ ('$\cdot - \circ - \cdot$'); G3-PCX, $\lambda = 2, \mu = 3, \sigma_\zeta = 1, \sigma_\eta = 1$('$\cdot \cdot + \cdot \cdot$').

comparable to DE (Figure 4.5(a)). With f_{lan}, $NP = 10^4$ was not enough for the algorithm to solve the problem without failure. However, the failure rate steadily approached zero, so it should reach zero with an increase in NP. In f_{rana} and the five-dimensional versions of f_{sal} and f_{whi}, the failure rate was again far from zero even with $NP = 10^4$. With function f_{rana}, changes in control parameter setup only had a clear effect on the convergence speed of the algorithm but not on the failure rate (Figure 4.6). The best choice for parameters seemed to be $\lambda = 2, \mu = 3, \sigma_\zeta = 0.1, \sigma_\eta = 0.1$, which led to the fastest convergence. With this setup, NFE_{slope} was gentle compared to DE and suggests that a considerably larger population size could be used with reasonable convergence speed. Still, increase in NP did not clearly improve the number of successful runs and thus the advantage gained for increasing NP even further is questionable. The situation was similar with the five-dimensional version of function f_{whi}, even though slightly better success rate was achieved using larger σ_ζ and σ_η. The five-dimensional f_{sal} was difficult

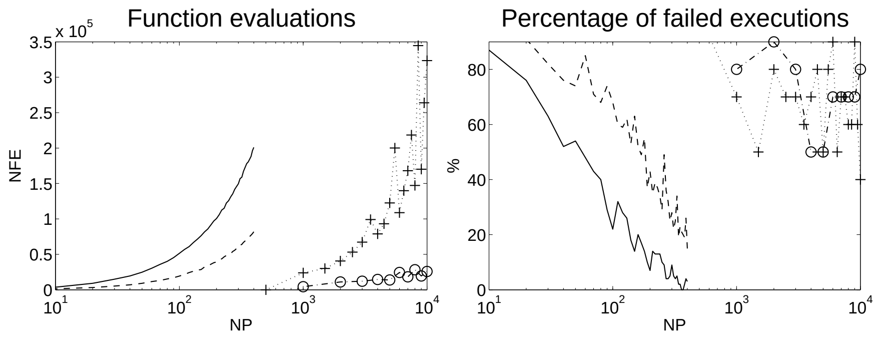


Figure 4.6: Average number of function evaluations required to reach the global optimum, and the percentage of failed runs for f_{rana} with $D = 2$ and $\varepsilon = 10^{-5}$. DE, $F = 0.9, CR = 0.2$ ('$-$'); DE, $F = 0.9, CR = 0.9$ ('$--$'); G3-PCX, $\lambda = 2, \mu = 3, \sigma_\zeta = 0.1, \sigma_\eta = 0.1$ ('$\cdot - \circ - \cdot$'); G3-PCX, $\lambda = 2, \mu = 3, \sigma_\zeta = 1, \sigma_\eta = 1$('$\cdot \cdot + \cdot \cdot$').

also for the G3-PCX algorithm (Figure 4.5(b)). Only the parameter setup of $\lambda = 2, \mu = 3, \sigma_\zeta = 1, \sigma_\eta = 1$ was able to produce solutions regularly. With this setup, G3-PCX found the optimum somewhat faster, but the failure rate was greater compared to the best case with DE. In all other tested cases, G3-PCX produced failures in almost 100% of the cases.

Most nonseparable functions benefited from large *CR* in convergence speed. The connection between smaller *CR* and a smaller minimum usable population size could be observed also in this kind of problems, with the exception of five-dimensional f_{whi}, where, $CR = 0.5$ produced a poor success rate. The increase in F again increased NFE_{slope} and decreased the minimum usable population in most cases. However, in some cases, the increase in *NP* could not patch the poor combination of small F and large *CR*, and thus the failure rates were high. In five-dimensional f_{sal}, the best result was acquired with $F = 0.2$.

Summary of the results

Neither of the tested algorithms had noticeable problems in solving the tested unimodal problems. The difference was in the convergence speed, and G3-PCX could achieve clearly superior performance in the function evaluations required to reach the optimum. Increase in problem dimensionality or increase in precision demand for the solution did not markedly change the relative situation.

The situation changed with multimodal problems. Even in some two-dimensional cases, the G3-PCX had difficulty in solving the problems reliably. A large population was needed to prevent premature convergence. When the dimensionality was increased to five, the algorithm was not able to reach the zero failure rate in any of the tested multimodal problems, regardless of the control parameter setup. In several cases, even the best control parameter setup was unable to find the global optimum in more than 50% of the runs. Similar difficulties for G3-PCX optimizing multimodal problems were reported in [4, 5]. In comparison, DE reached a zero or close to zero (in f_{sal} and f_{rana}) failure rate in all tested cases. DE performed especially well in separable multimodal problems with small *CR*.

In most of the tested cases, DE was not very sensitive to the control parameter setup, and could solve problems reasonably well even with roughly set parameters. The use of a large parameter F value with DE generally increased the robustness of the algorithm, especially if the initialization was skewed, but in most cases, the price to pay was decreased convergence speed in the form of steeper NFE_{slope}. Parameter *CR* controls the degree of rotation invariance of the search, making use of a small value desirable with separable problems. With nonseparable problems, large *CR* enabled the search to proceed freely, regardless of the axis directions, again increasing convergence speed. The *CR* parameter appeared also to affect the minimum usable population size: a small value generally enabled the algorithm to solve the problem reliably with smaller *NP*. The effect was especially noticeable when using the combination of small F and large *CR*, which often resulted in fast but unreliable convergence. The effect can be explained by the additional diversity provided by using the crossover. As small F enabled only small jumps, the crossover was able to increase the pool of potential solutions the algorithm was able to generate.

For the G3-PCX algorithm, parameters λ and μ appeared to have rather a small effect on its ability to find the global optimum, but increasing the value of either notably decreased the convergence speed. Parameters σ_ζ and σ_η had a more notable effect on failure rate. In many problems, using the values of 1 for either instead of 0.1 decreased the failure rate noticeably, but the number of function evaluations required to solve the problem was also typically multiplied. A limited study with f_{ras} was also done to see if even larger σ_ζ and σ_η would improve the performance of G3-PCX algorithm. Also, a setup with larger values for all control parameters was tested. None of these cases suggested improved performance.

4.3.5 Discussion and conclusions

The results show clearly that G3-PCX excels in unimodal problems due to the greediness of the approach, which turns against the algorithm in multimodal cases. DE, on the other hand, can exploit the separability of the problems efficiently when small *CR* is used and is thus very efficient in separable multimodal problems. This demonstrates well the critical effect the selection of test problems has in experimental studies comparing algorithms. Selecting a test setup including functions sharing common features favors algorithms that exploit those features. If the reasons for good or poor performance are not carefully studied and documented, the results of such studies can be misleading. Using a test set with well understood functions with varying features increases the reliability of the conclusions drawn from the experimental data.

However, the selection of appropriate test functions is not the only difficulty of doing a quality comparisons between different algorithms. Let us consider the results of the current study from a wider perspective: can it be really said on the basis of the presented results that DE outperforms G3-PCX in multimodal problems? The answer is actually yes and no at the same time. The results do indeed demonstrate that the particularly greedy version of G3-PCX used in this study and in [35] has a poor performance in multimodal functions. The reason for this is simply that the offspring are generated around the best population member. This means that basically the algorithm relies on the initial population to offer good enough coverage of the function surface to be able to estimate the general area of global optimum, and then simply performs an extended local search (in [91] G3-PCX is classified as an XLS algorithm) on this area to locate the actual global optimum. This explains the very large population sizes required by the G3-PCX. While such a strategy is efficient in functions with a low number of optima, it becomes unreliable in functions with a high number of optima and high dimensionality. Parameters σ_ζ and σ_η basically control the size of the area in which the extended local search is performed. A similar effect could be gained in DE by replacing the DE/rand/1/bin with DE/1/best/bin [125], which also generates trials around the best population member and disabling the crossover by using value $CR = 1$. Now instead of σ_ζ and σ_η in G3-PCX, the parameter F controls the area in which the extended local search is performed. Using a small F would result in a similar very greedy DE algorithm as the tested version of G3-PCX.

Similarly, the definition in [35] allows a less greedy version of the G3-PCX, which performs the extended local search in parallel around several population members. Such usage for the PCX-operator is suggested in [37] in context of multiobjective-optimization in conjunction with the elitist non-dominated sorting GA. Dividing the search efforts to several areas would of course decrease the convergence speed of the algorithm in unimodal functions, but it would also increase the reliability in multimodal problems. It is possible that by using proper versions and parameter combinations for both methods, comparable performance in unimodal and nonseparable multimodal functions could be achieved. The exception for this are the separable multimodal functions, which G3-PCX can not exploit. The study thus demonstrates also another difficulty in comparing algorithms: the selection of the used algorithm version and the parameter settings. Even with a well defined test setup including varied functions and seemingly exhaustive parameter study, the results are inconclusive for deciding the general comparable performance between the G3-PCX and DE algorithms, as using different versions of the algorithms could potentially change the situation dramatically. However, such an extended comparison of different algorithm versions is beyond the scope of this study.

4.4 Test functions for multimodal optimization

The contents of this section have been originally published by the author in [137, 138]. While a wide variety of global optimization test problems and problem generators exist, the available selection of suitable problems for testing multimodal optimization algorithms is notably limited. Most multimodal optimization methods have been evaluated using only one- or two-dimensional multimodal test functions. Furthermore, these functions are often defined in a way which does not allow the function to be changed in terms of the characteristics of multimodal landscapes. For example, for the Shubert function used in [84], as the number of dimensions increases, the number of global optima grows exponentially ($D \cdot 3^D$, where D is the number of dimensions). There is no way to control the number of optima, nor how they are distributed. Additionally, the problem is separable, and the global optima are positioned at regular intervals in the search space, both being easily exploitable features. In short, using solely the currently available selection of test functions is inadequate for proper analysis of the characteristics of different multimodal optimization algorithms. For this reason, a software framework has been developed as a part of this work for generating tunable multimodal test functions. The aim is to provide a general and easily expandable environment for testing different methods of multimodal optimization. The term test function *generator* is used in this work for the resulting software module, although the framework actually contains several function families with different characteristics. The current software version is 1.1 and includes cosine, quadratic, hump and common families, but the framework is easily expandable, and new families may be added.

4.4.1 Existing function generators

Several function generators that are able to generate multimodal functions with real parameter spaces have been previously presented in the literature. The most interesting among such approaches are analyzed below. To the author's knowledge, no previous versatile environment designed especially for producing problems for multimodal optimization and measuring the performance of multimodal optimization algorithms exists.

The DF1 [107, 106] and Moving Peaks [13, 14] generators focus on generating dynamic multimodal landscapes that change over time. The generators allow the construction of landscapes with a desired number of optima and dimensionality. The optima locations, heights and slopes are tunable and can change dynamically. While in principle the generators could be used to produce functions for evaluating multimodal optimizers, the current implementations are designed only to provide information from the global optimum. Thus evaluating the multimodal performance would be difficult and the generators are ill suited for the task. Additionally the generators do not allow the shape of the optima to be changed (except for their height and slope).

Gaviano et al. [48] have generated differentiable multimodal functions with a single global optimum by distorting convex functions using polynomials. Macnish [92] proposes a fractal landscape generator capable of generating complex surfaces by simulating random meteor impacts. The inability to produce functions with multiple global optima and to offer exact information of the local optima are obvious limitations for the usability of both methods in the context of multimodal optimization. The constrained test cases generator [102, 103], on the other hand, allows multiple global optima to be generated. The method generates function landscapes by dividing the search space into regions and constructing a simple unimodal function for each region. The main feature of the approach is that it allows the definition of constraint functions for each of these regions. Constrained optimization, however, is outside the scope of this thesis. When the constraint functions are ignored, however, the landscapes are very simple and of little use in evaluating multimodal optimization approaches.

The two most promising methods for generating multimodal test functions are the Max Set of Gaussians (MSG) landscape generator presented by Gallagher and Yuan [45], and the method for generating composition test functions by Liang et al. [86]. The MSG combines several independent peaks to form the function landscape, while the composition landscape is formed by combining several standard benchmark functions. Both methods allow several global optima to be generated and are able to provide information of their locations, which allows usage in the context of multimodal optimization. Further discussion of both methods follows in Section 4.4.3.

4.4.2 Cosine family

The cosine family allows functions with a high number of optima to be generated with low computational cost. The degree of regularity and separability may be controlled by rotating and stretching the functions. Two cosine curves are sampled together, one of which defines global minima and the other adds local minima. The basic internal structure is regular: all minima are of similar size and shape and located in rows of similar distance from each other (see Figure 4.7(a)). The function family is defined by

$$f_{cos}(\vec{y}) = \frac{\sum_{i=1}^{D} -cos((G_i-1)2\pi y_i) - \alpha \cdot cos((G_i-1)2\pi L_i y_i)}{2D} \tag{4.17}$$

where $\vec{y} \in [0,1]^D$, the parameters $\vec{G}$ and $\vec{L}$ are vectors of positive integers which define the number of global and local minima for each dimension, and $\alpha \in]0,1]$ defines the amplitude of the sampling function (depth of the local minima).

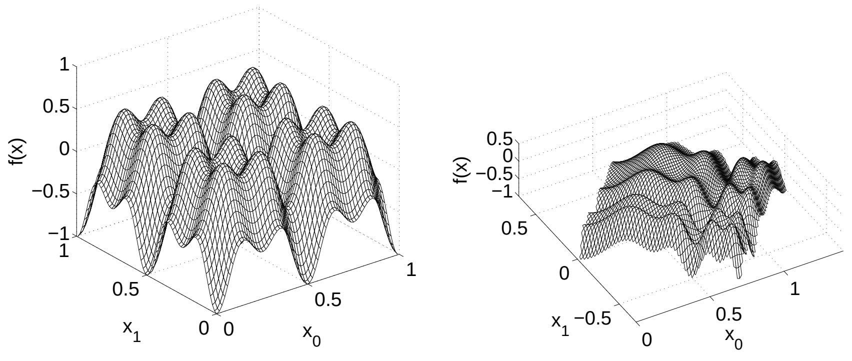


Figure 4.7: Example figures of cosine family functions using parameter values $\alpha = 0.8$, $\vec{G} = [3,3]$, $\vec{L} = [2,2]$ (9 global and 16 local minima). Additionally, $\vec{P_1} = [0, 0.1, 0.2, 0.5, 1]$ and $\vec{P_2} = [0, 0.5, 0.8, 0.9, 1]$ are used for (b).

The framework allows the function to be rotated to a random angle and uses Bezier curves [10, 11] to stretch each dimension independently to decrease the regularity (see Figure 4.7(b)). To calculate the function value for input vector $\vec{x}$, $\vec{x}$ is first mapped to $\vec{y}$. The mapping proceeds in two steps, where the first step is to calculate $\vec{b}$, which is the rotated point corresponding to $\vec{x}$

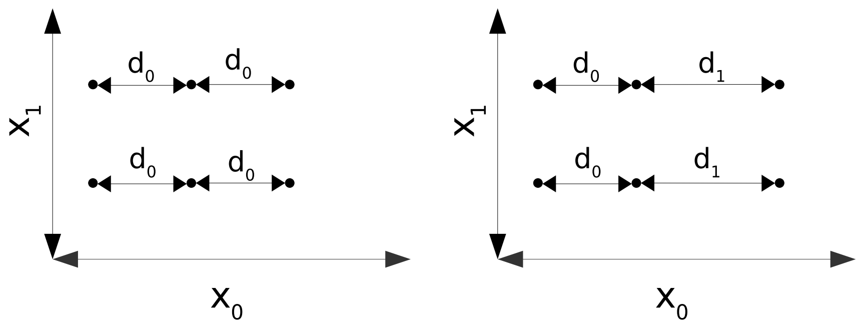
$$\vec{b} = \mathbf{O}\vec{x} \tag{4.18}$$

where the matrix $\mathbf{O} = [\vec{o}_1, \ldots, \vec{o}_D]$ is a randomly generated angle preserving orthogonal linear transformation, as described in [59]. The domain of $\vec{x}$ (the search space) is the D-dimensional unit hypercube rotated with $\mathbf{O}^T$. Then $\vec{b}$ is mapped to $\vec{y}$ by applying a Bezier formula

$$y_i = \sum_{j=0}^{\nu_i} \binom{\nu_i}{j} P_{i,j}(1-b_i)^{\nu_i - j} {b_i}^j, i = 1, \ldots, D \tag{4.19}$$

where ν_i is the degree of the Bezier curve for dimension i and defines the number of the control points used. $\vec{P_i}$ are the control point vectors defined so that $P_{i,0}$ and P_{i,ν_i} correspond to the lower and upper bound of y_i, and the values between the bounds are strictly increasing.

The Bezier stretching will decrease the regularity of the function, but generally not completely eliminate it, because the function is regular along directions defined by $\mathbf{O}$, as demonstrated in Figure 4.8. The degree of regularity can be roughly measured by considering the minimum number of global minima points required to have a set which contains all possible differentials to jump from a neighboring minimum to the next in the axis directions. For completely regular functions, only D+1 points are required (as long as there is more than one minimum along each dimension). For stretched functions, the required number is $\sum_{i=1}^{D}(G_i - 1) + 1$ out of the total number of global minima $\prod_{i=1}^{D} G_i$. So the degree of regularity increases along with the number of dimensions. Bezier stretching also affects the shape and size of the minima, increasing the differences in their AOA.



(a) Regular (b) Stretched

Figure 4.8: The effect of stretching on the regularity of a function. The points are minima locations.

The number of minima increases exponentially along with the number of dimensions. The number of local minima which are not global is $\prod_{i=1}^{D}[G_i + (G_i - 1)(L_i - 1)]$. For each dimension, two of the outermost minima will always be located on the constraints. This means that if any of the elements of $\vec{G}$ are less than 3, every minimum will be located on at least one constraint. In the unstretched case, each constraint on which the minimum sits, halves the area of attraction for that minimum compared to a minimum with one less constraint. The fraction of the AOA from the full possible area is thus $1/2^{D-l}$, where $l = 0, 1, \ldots, D$ describes the number of constraints the minimum sits on. $l = 0$ means a minimum located on no constraint (full possible AOA) and $l = D$ is a corner minimum with minimum AOA for dimension D. For methods that rely heavily on the initial points, locating the minima on corners becomes harder with increasing dimensionality, unless the information that the minima are located on the constraints is exploited.

Parameter α affects the depth of the local minima. Increasing the value makes the minima deeper, also increasing their area of attraction, and thus slightly increasing the difficulty of the problem. Examples of two-dimensional cosine family functions are presented in Figure 4.7.

4.4.3 Quadratic family

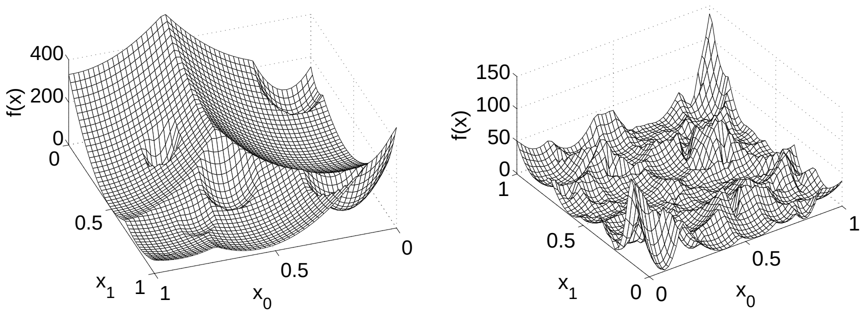
The quadratic family is used to generate completely irregular landscapes, and allows the number of minima to be defined independently of the number of dimensions (see Figure 4.9). The user may select any number of global and local minima. The function is created by combining several minima generated independently. Each minimum is described as a D dimensional general quadratic form, where a symmetric matrix $\mathbf{C}$ defines the shape. The functions in a quadratic family need not be stretched or rotated, because no additional benefit would be gained as they are already irregular functions. However, axis-aligned (hyper) ellipsoidal minima may be randomly rotated by rotating matrix $\mathbf{C}$ as follows:

$$\mathbf{B} = \mathbf{OCO}^T \tag{4.20}$$

The functions are calculated by

$$f_{quad}(\vec{x}) = \min_{i=1,2,\ldots,n} \left((\vec{x} - \vec{q}_i)^T \mathbf{B}_i^{-1} (\vec{x} - \vec{q}_i) + v_i \right) \tag{4.21}$$

where $\vec{x} \in [0,1]^D$, $\vec{q}_i$ defines the location, and v_i the fitness value of a minimum point for the i'th minimum. n is the number of minima.



(a) Spherical, no local minima (b) Rotated ellipsoidal with local minima

Figure 4.9: Examples of quadratic family functions. The minimum Euclidean distance between the global minima is set to 0.01, and the shape range for all minima to [0.003,0.03]. (a) has 10 global spherical minima, while (b) has 10 global and 100 local rotated ellipsoidal minima so that the local minima points have fitness values in the range [-0.95,-0.15]. The globally minimal value is always -1.

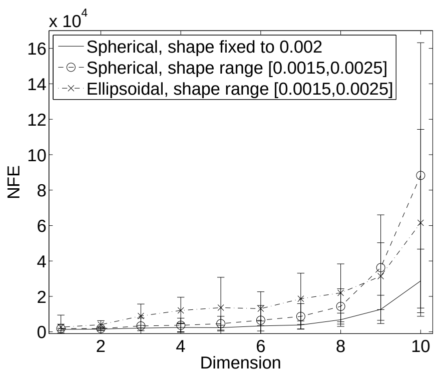
The placement of minima is chosen randomly, although the minimum Euclidean distance between global minima may be defined. The module makes certain that no minimum is completely engulfed by another, deeper minimum. The user may also define the shape of minima, which may be spherical, ellipsoidal or randomly rotated ellipsoidal. This selection is used for all generated minima. The shape range for global and local minima may be defined independently, and the shapes are generated by using uniform random numbers from this shape range for each dimension in creating matrix $\mathbf{C}$. The user may also define the range for the fitness values of local minima points. Figure 4.9 presents examples of functions from the quadratic family.

The locations of minima are random, and the AOAs of minima can be inside each other, the sizes of the areas of attraction for different minima will vary greatly. Using fewer or shallower local minima will naturally leave more room for the global minima. Forcing a longer minimum Euclidean distance between the global minima will also leave more area for each minimum. As the dimensionality increases, the differences in the shape parameters will have an exponentially increasing effect on the AOAs. Small differences in shape parameter values can lead to large differences in the relative sizes of the AOAs in high dimensions.

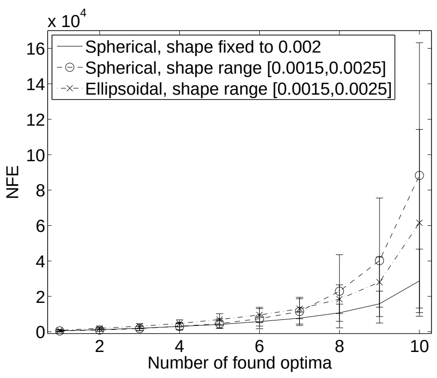
To demonstrate this, the RSGD algorithm was run on three different sets of quadratic functions, with $D = 1, 2, \ldots, 10$. The average performance of RSGD is a decent estimate of the relative sizes of AOAs, because to locate a minimum, the random starting point must be located in the AOA of that particular minimum. If the sizes differ significantly, the larger ones will draw more points, slowing down the process of locating the minima with a smaller AOA. Figure 4.10 displays the results of the runs. The first set of functions has spherical minima with identical sizes. The second also has spherical minima and the third ellipsoidal minima, both allowing the shape range to change by $\pm 50\%$. As can be seen in the figure, the required NFE, as well as the standard deviation increase notably more slowly along with the increase in dimensionality for the first set compared to the other two. This is to be expected, because the identical shape eliminates the differences in AOA caused by shape. The ellipsoidal shape is the most difficult to solve when the dimensionality is low, but varying spherical shape becomes more difficult when $D > 8$. This is logical, because for a spherical shape only one random value is generated, which is then used in all dimensions. If the value is small, it will affect all dimensions. For an ellipsoidal shape, each dimension will get a different random value, and the AOA will on average vary less as more values are generated in higher dimensional cases. The slower performance on the low dimensional ellipsoidal set can be explained by the fact that the gradient descent tends to oscillate on nonspherical shapes and, thus needs more line searches to find the minimum point compared to spherical shapes, where the gradient points directly to the minimum point.

Comparison to other methods

The MSG [45] generator resembles the quadratic family, as peaks are similarly produced independently and the dominant peak is used to define the function value at point $\vec{x}$. MSG uses a Gaussian density function to define the peaks. For the quadratic family, a general quadratic form was selected (which is similar to the exponent part of the Gaussian density function, when the constant is removed) to describe the minima. As a result, the landscapes generated by MSG have more localized optima shapes, as well as large almost flat regions compared to the ones generated by the quadratic family. In theory, both



(a) Function evaluations required to find all minima with precision 0.0001



(b) Function evaluations required to find the i'th minimum in the 10-dimensional case

Figure 4.10: Performance of RSGD on different quadratic functions with 10 global and no local minima. The minimum Euclidean distance between the global minima is set to 0.1. The function evaluations are averages from 100 independent runs, and the figures include standard deviations. For each run, a different random seed is used in generating the function.

approaches allow landscapes with no completely flat areas to be generated. However, a potential problem with MSG is numerical precision: away from an optimum, it is possible that the search space contains areas which seem flat because of limited numeric precision. When using the quadratic form, this is not an issue.

The composition landscape proposed in [86] is formed by combining several basic benchmark functions. When using solely unimodal basic functions, the produced landscape resembles the one generated by the quadratic family. The number and location of both global and local optima may be controlled independently of the number of dimensions. Additionally, the basic functions may be rotated, although the only unimodal basic function currently included is the symmetric sphere, on which rotation has no effect. The proposed model allows each basic function to be scaled independently, but each dimension is scaled equally and the shape of the function remains unchanged. Including ellipsoidal unimodal basic functions and adding a dimensional scaling would allow the composition functions to mimic the rotated ellipsoidal optima shapes of the quadratic family. However, the most interesting feature of the composition principle is that it allows the landscape to be constructed of different basic components, which may already be multimodal. While this means that the number of optima would no longer be independent of the number of dimensions, and the number and locations of local optima would be harder to define, it would allow the definition of changing landscapes with a lot of optima, which requires less computational effort compared to the quadratic family. Thus adding a new family adopting the composition principle might be an interesting option in future development of the generator framework.

4.4.4 Hump family

The hump family implements the generic hump functions family proposed by Singh and Deb [151], see Figure 4.11. Like the quadratic family, the hump functions allow irregular landscapes to be generated and the number of minima to be defined independently of the dimensionality. The placement of minima is chosen randomly. Each minimum is defined by

$$f_h(\vec{y}) = \begin{cases} v_i \left[1 - \left(\frac{d(\vec{y},i)}{r_i^{rad}}\right)^{\beta_i}\right], & \text{if } d(\vec{y},i) \leq r_i^{rad} \\ 0, & \text{otherwise} \end{cases} \tag{4.22}$$

where: $\vec{y} \in [0,1]^D$, v_i is the fitness of the i'th minimum. $d(\vec{y},i)$ is the Euclidean distance between $\vec{y}$ and the center of the i'th minimum. $r_i^{rad} \in [0.001, \infty)$ defines the basin radius and $\beta_i \in [0.001, 1]$ the shape of the i'th minimum slope.

Each minimum in hump functions has a fixed radius value, and all values outside the radius value are set to a constant 0. Thus, the function surface is flat between the minima. This is problematic if small radii are used, because the result is a needle-in-a-haystack problem, where the majority of the search space is flat, including only some thin holes. Especially for methods relying on gradient information, the flat surface makes the gradients unusable. However, the hump family offers better control over the AOAs of minima compared to the quadratic family, because the AOAs do not intersect. Furthermore, when reasonably large radii for the minima are used, the hump family is suitable for testing the ability of an algorithm to handle flat areas on a function surface. The minima are always spherical in shape, but the hump functions can be rotated and stretched similarly to cosine family functions to change the shape.

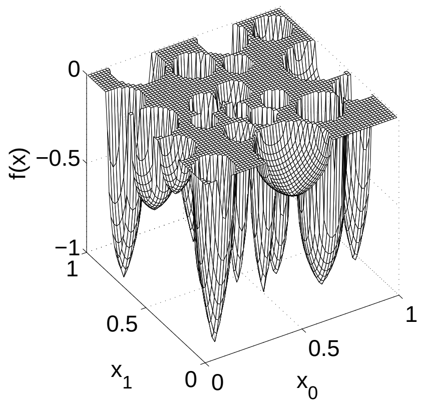


Figure 4.11: Example of a hump family function. The function has 10 global and local minima so that the local minima points have fitness values in the range [-0.5,-0.15]. The globally minimal value is always -1. The radii range for all minima is set to [0.05,0.2] and the shape parameter β_i is in the range [0.2, 0.5].

4.4.5 Common family

The common family collects some well known multimodal test problems from the literature and implements them inside the module framework. The module allows similar rotation and Bezier stretching as used in the cosine family for all implemented functions in the common family. At the moment, eight different functions having multiple global minima are implemented. The implemented functions are:

The Branin function [101], Figure 4.12(a)

$$f_{bra}(\vec{y}) = \left(y_2 - \frac{5.1}{4\pi^2}y_1^2 + \frac{5}{\pi}y_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(y_1) + 10 \tag{4.23}$$

where $y_1 \in [-5, 10]$, $y_2 \in [0, 15]$, has three irregularly spaced global minima.

The Himmelblau function [7], Figure 4.12(b)

$$f_{him}(\vec{y}) = (y_1^2 + y_2 - 11)^2 + (y_1 + y_2^2 - 7)^2 \tag{4.24}$$

where $\vec{y} \in [-6, 6]^2$, has four irregularly spaced global minima.

The Shubert function [84], Figure 4.12(c)

$$f_{shu}(\vec{y}) = \sum_{i=1}^{5} i\cos((i+1)y_1 + i) \cdot \sum_{i=1}^{5} i\cos((i+1)y_2 + i) \tag{4.25}$$

where $\vec{y} \in [-10, 10]^2$, has 18 global and 742 local minima. The global minima are situated in nine groups of two closely situated minima. The groups form a shape of 3 by 3 square. The groups as well as the minima inside the groups are regularly spaced.

The Six-hump camel back function [101], [169], Figure 4.12(d)

$$f_{shcb}(\vec{y}) = \left(4 - 2.1y_1^2 + \frac{y_1^4}{3}\right)y_1^2 + y_1y_2 + (-4 + 4y_2^2)y_2^2 \tag{4.26}$$

where $y_1 \in [-1.9, 1.9]$, $y_2 \in [-1.1, 1.1]$, has two global and four local partially regular minima.

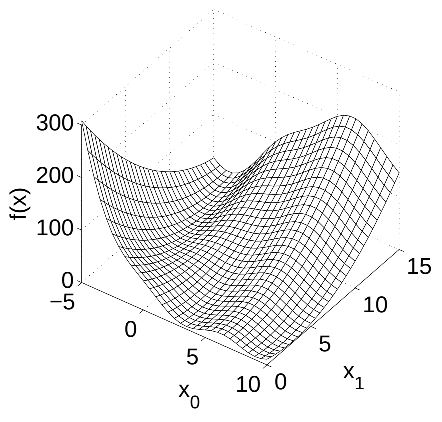
The Vincent function [150], Figure 4.12(e)

$$f_{vin}(\vec{y}) = -\frac{1}{D}\sum_{i=1}^{D} \sin 10 \cdot \log(y_i) \tag{4.27}$$

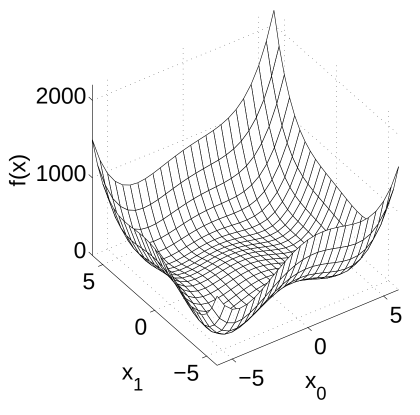
where $\vec{y} \in [0.25, 10]^D$, has 6^D global minima. The function is partially regular, like a Bezier stretched function of the cosine family. The differences between the minima are not random but increase along the value of y.

The modified Rastrigin function, Figure 4.12(f)

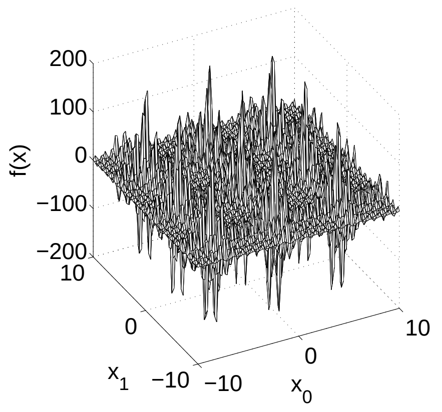
$$f_{ras}(\vec{y}) = 20 + \sum_{i=1}^{2}\left(y_i^2 + 10\cos(2\pi y_i)\right) \tag{4.28}$$



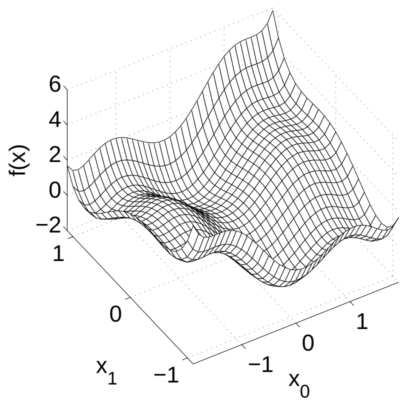
(a) Branin



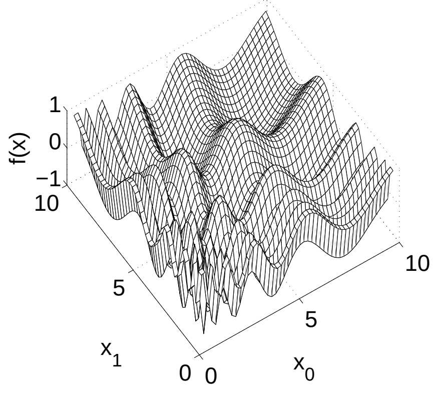
(b) Himmelblau



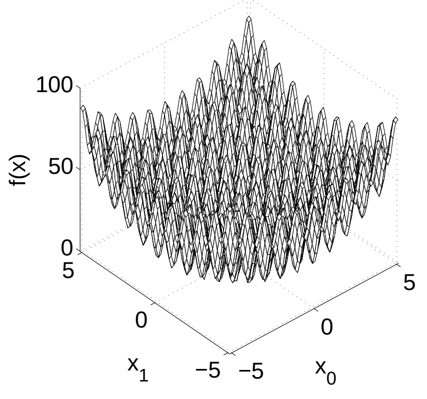
(c) Shubert



(d) Six-hump camel back



(e) Vincent



(f) Modified Rastrigin

Figure 4.12: Common family functions having multiple global optima

where $\vec{y} \in [-5.12, 5.12]^2$, is a version of the Rastrigin function modified to contain more than one global minimum. It has four regularly spaced global minima and 96 local minima.

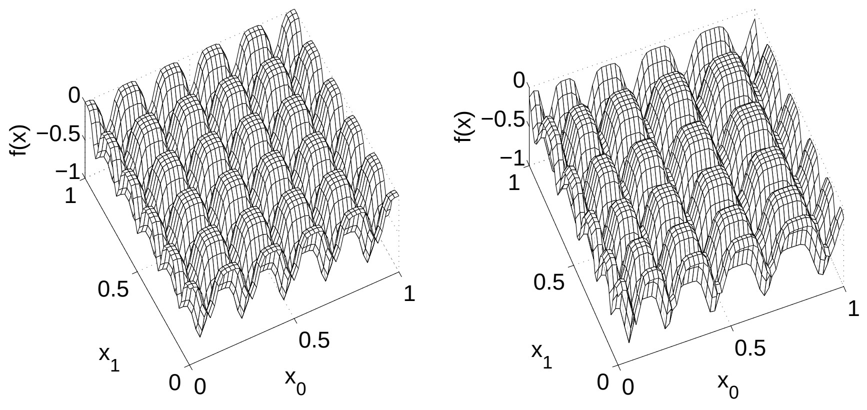
Deb's 1st

$$f_{deb1}(\vec{y}) = -\frac{1}{D}\sum_{i=1}^{D} \sin^6(5\pi y_i) \tag{4.29}$$

and 3rd functions [7], Figure 4.13

$$f_{deb3}(\vec{y}) = -\frac{1}{D}\sum_{i=1}^{D} \sin^6\left(5\pi\left(y_i^{\frac{3}{4}} - 0.05\right)\right) \tag{4.30}$$

where $\vec{y} \in [0, 1]^D$, have 5^D global minima. The functions are distinguished by the distribution of the optima. While f_{deb1} is regular, the distances between the optima in f_{deb3} decrease with the value of $\vec{y}$, making the function partially regular.



(a) Deb's 1st (b) Deb's 3rd

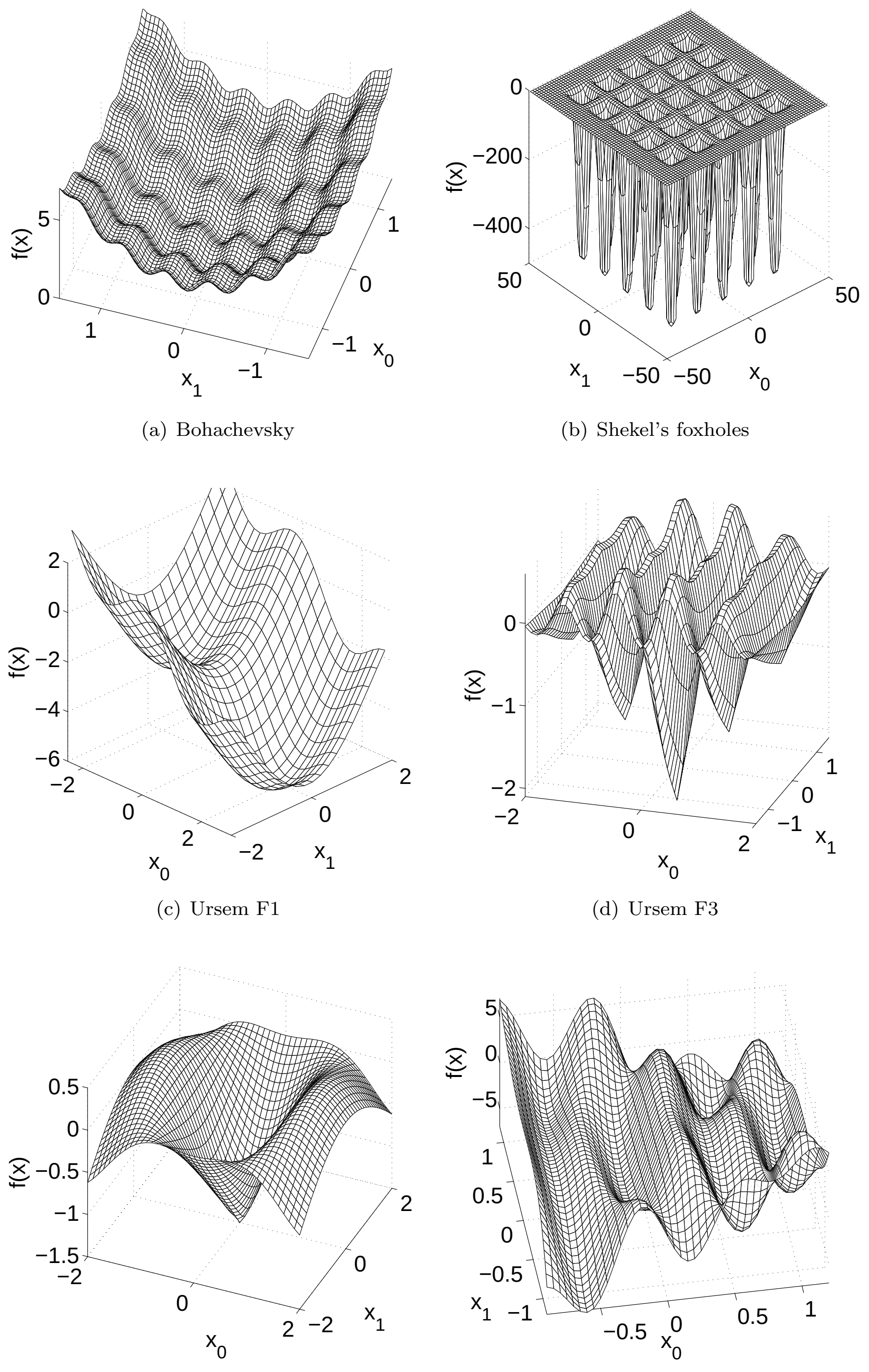
Figure 4.13: Deb's 1st and 3rd functions

Additionally, the family implements another set of eight functions with a single global minimum and a set of local minima to test the ability of algorithms to locate and keep good local minima in addition to the global ones. The functions are taken from [163, 169, 50] and include the following:

The Bohachevsky function, Figure 4.14(a)

$$f_{boh}(\vec{y}) = y_1^2 + 2y_2^2 - 0.3\cos(3\pi y_1) - 0.4\cos(4\pi y_2) + 0.7 \tag{4.31}$$

where $\vec{y} \in [-50, 50]^2$, has one global and several regularly spaced local minima. Among the local minima, only the eight with the best fitness value are used to decide success of an algorithm in locating the local minima. They are positioned in a 3 by 3 grid around the global minimum.



(a) Bohachevsky

(b) Shekel's foxholes

(c) Ursem F1

(d) Ursem F3

(e) Ursem F4

(f) Ursem waves

Figure 4.14: Common family functions having a single global optimum

Shekel's foxholes function, Figure 4.14(b)

$$f_{she}(\vec{y}) = -500 + \frac{1}{0.002 + \sum_{j=1}^{25}(j + \sum_{i=1}^{2}(y_i - a_{ij})^6)^{-1}} \tag{4.32}$$

where

$$\mathbf{A} = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & \dots & 32 & 32 & 32 \end{pmatrix}$$

and $\vec{y} \in [-65.536, 65.536]^2$, has one global and 24 regularly spaced local minima positioned in a 5 by 5 grid. The global minimum is located at the corner of the grid.

The Ursem F1 function, Figure 4.14(c)

$$f_{ur1}(\vec{y}) = -\sin(2y_1 - 0.5\pi) - 3\cos(y_2) - 0.5y_1 \tag{4.33}$$

where $y_1 \in [-2.5, 3]$, $y_2 \in [-2, 2]$, has one global and one local minimum.

The Ursem F3 function, Figure 4.14(d)

$$f_{ur3}(\vec{y}) = -\sin(2.2\pi y_1 + 0.5\pi) \cdot \frac{2 - |y_2|}{2} \cdot \frac{3 - |y_1|}{2} - \sin(0.5\pi y_2^2 + 0.5\pi) \cdot \frac{2 - |y_2|}{2} \cdot \frac{2 - |y_1|}{2} \tag{4.34}$$

where $y_1 \in [-2, 2]$, $y_2 \in [-1.5, 1.5]$, has one global and four regularly spaced local minima positioned in a direct line, so that the global minimum is in the middle.

The Ursem F4 function, Figure 4.14(e)

$$f_{ur4}(\vec{y}) = -3\sin(0.5\pi y_1 + 0.5\pi) \cdot \frac{2 - \sqrt{y_1^2 + y_2^2}}{4} \tag{4.35}$$

where $\vec{y} \in [-2, 2]^2$, has one global minimum positioned at the middle and four local minima at the corners of the search space.

The Ursem waves function, Figure 4.14(f)

$$f_{urw}(\vec{y}) = -(0.3y_1)^3 + (y_2^2 - 4.5y_2^2)y_1y_2 + 4.7\cos(3y_1 - y_2^2(2 + y_1))\sin(2.5\pi y_1) \tag{4.36}$$

where $y_1 \in [-0.9, 1.2]$, $y_2 \in [-1.2, 1.2]$, has one global and nine irregularly spaced local minima.

The final two included functions are the Ripple

$$f_{rip}(\vec{y}) = \sum_{i=1}^{2} -e^{-2\ln 2(\frac{y_i - 0.1}{0.8})^2}(\sin^6(5\pi y_i) + 0.1\cos^2(500\pi y_i)) \tag{4.37}$$

and Ripple25 functions, Figure 4.15

$$f_{r25}(\vec{y}) = \sum_{i=1}^{2} -e^{-2\ln 2(\frac{y_i - 0.1}{0.8})^2}(\sin^6(5\pi y_i)) \tag{4.38}$$

where $\vec{y} \in [0, 1]^2$. The Ripple has one global and 252004 local minima. The global form of the function consists of 25 holes, which form a 5 by 5 regular grid, as shown by Figure 4.15. Additionally, the whole function surface is full of small ripples caused by the high frequency cosine function, which creates a large number of small local minima. Among the minima, the best fitness of the 25 holes are used to decide success of an algorithm in locating the minima. The global minimum is located at the corner of the grid. The Ripple25 contains the global form of the Ripple function without the ripple. Ripple and Ripple25 can be used as a pair to compare the ability of an algorithm to handle local noise.

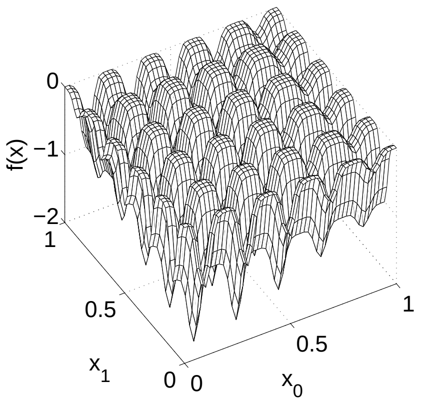


Figure 4.15: Ripple25 function

4.4.6 Problem features and the proposed framework

Some important features of multimodal functions were listed in Section 4.1. This section summarizes the relation of different function families to each of the features.

Dimensionality and the number of optima

The cosine, quadratic and hump families allow the dimensionality to be scaled freely. Some common family functions have a fixed dimensionality. The quadratic and hump families allow the number of local and global optima to be set precisely and independently of the number of dimensions. For the cosine and common family functions, the number of optima increases exponentially along with the number of dimensions.

Relative area of attraction sizes

The hump family allows the relative AOAs to be directly controlled. The quadratic family allows similar but less precise control, due the fact that the AOAs of different optima may intersect. In the cosine and common families, the relative AOA sizes can be altered by stretching the functions.

Relative fitness of optima

Cosine family functions have groups of local optima with similar fitness, whose depth can be controlled. Quadratic and hump families allow the definition of a range in which the fitness of each local optimum is randomly generated. For the common family, each individual function defines the fitnesses, and they cannot be modified.

Separability

Quadratic and hump families have high epistasis because they are always nonseparable. The epistasis of the cosine and common family functions can be controlled by rotating the functions.

Directional bias and regularity

The quadratic and hump family functions are irregular and do not embody directional biases. The cosine family functions always have directional bias. Their degree of regularity can be controlled between regular and partially regular by using stretching and controlling the number and positions of the optima. The common family functions can also be stretched, but some of them are already irregular or partially regular by definition.

Symmetry

The quadratic and hump families are not symmetric. The cosine family functions for which G_i and L_i are defined as equal for all dimensions are completely symmetric in their basic form. This means that they are symmetric in regard to all the $D!$ possible symmetry equivalences. Stretching and defining varying G_i or L_i of optima for different dimensions can be used to control the amount of symmetry: removing symmetry from n dimensions leaves $(D - n)!$ symmetry equivalences in regard to which the function is symmetric. Some of the common family functions are also symmetric. Using rotation removes the symmetry. The proposed framework does not currently contain rotationally symmetric functions.

Continuity and differentiability

All the functions within the framework are continuous and allow a numerical approximation of the first order derivative to be generated at any point, even if the derivative does not exist in the analytical sense. The framework has not been designed to be used with methods which require analytical first or higher order derivatives. The hump family functions and some of the functions in the common family contain areas of flat fitness, which pose a special challenge for methods using derivatives to direct the search.

Global structure

Although the goal of multimodal optimization is to locate multiple optima, averaging and copying can be used to some extent in exploiting the structure in certain functions.

The cosine family functions in their basic form will often have one global optimum in the middle of the search space and several optima which have equal values for all or most parameters. Rotation and stretching can be used to remove the equal parameter values from most of the optima, but one global optimum will always be located at the origin. The quadratic and hump families generate the optima locations randomly, and thus planned exploitation of global structure is not possible. None of the families currently allow the design of specific types of global structure, like specifically planted valleys or concentrations of optima.

Optima on constraints

As the optima locations in the quadratic and hump families are randomly selected, they will be rarely positioned on the constraints. However, depending on the selection of the $\vec{G}$ and $\vec{L}$ parameters of the cosine family, a significant portion or even all of the global optima may be located on the constraints. For methods using solely the internal constraint handling of the module, the search space seems to continue past the constraints, and the constraints become irrelevant. However, the module provides also exact constraint information if required by an algorithm, and thus allows it to be exploited.

4.4.7 Implementation and features

The software has been written in the C programming language obeying the ANSI standard. C was selected over C++ for maximum compatibility to optimization algorithms written in C or C++. C as a compiled language is also fast compared to most other popular programming languages, and compilers are freely available. From the outset, the idea has been to develop a general framework for evaluating multimodal optimization algorithms. Thus the structure of the software has been designed to allow easy addition of new function families to the module. Figure 4.16 presents the general structure of the generator module as an UML class diagram. While ANSI C does not directly support object-oriented concepts, they can be closely imitated. Thus the term class will be used when referring to parts of the software.

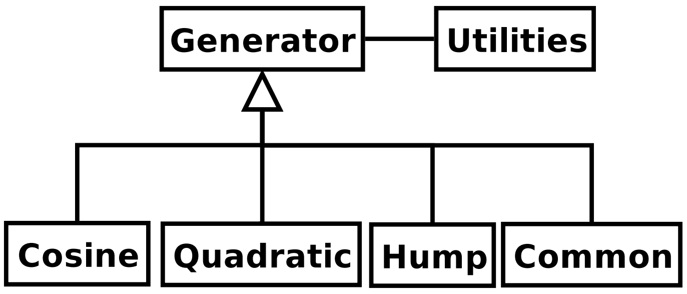


Figure 4.16: UML class diagram of the software structure

Inheritance is imitated by defining a set of function pointers in the generator class, which represent the functions provided by the public interface. Each class representing the different function families must provide an implementation for each of the interface functions and an initialization function which connects the function pointers to actual implementations. Adding a new function family thus requires adding a definition of the new initialization function to the generator class, which would not be necessary for languages supporting inheritance through dynamic binding. However, this is seen as a minor inconvenience, as in all other respects implementation of the new family can be done completely within its own compile unit. All function calls from the public interface are first handled by the generator class, which is responsible for handling tasks collective to all families, and the call is then redirected to the correct family as necessary. Thus each new family needs to contain only implementation directly related to the family. The utilities class provides implementation of functionalities used by multiple families, like rotation, stretching and constraint handling, which are readily available to new families.

The module is used by simply including the header file containing the public interface to optimization software. The user can then directly use all features provided by the generator. Function configuration is performed using text files. Each function instance is explicitly specified by an initialization file, except for the seed to the internal random number generator for functions that use randomization. The user may provide the seed through the call of initialization function. The same function is always generated with the same seed. Thus it is easy to define exact test sets by providing the initialization files and the information of used seeds. These features make the generator easy to use.

The module framework includes an internal constraint handling method to keep the solutions within a given range. The constraints are handled by mirroring the violating value back from the violated boundary by the amount of violation. This makes the function space look continuous for the optimization approach during the run, because any minimum which is located on the boundary looks symmetrical, although in reality the value is calculated in a mirrored point inside the boundaries. If required, the internal constraint handling can be ignored, and the linear constraint functions can be acquired in analytic form.

To help in evaluating the quality of a solution provided by an algorithm, the module offers a method for deciding how many different globally minimal solutions a given population contains with a required precision and their exact locations. A similar method is provided for the quadratic, hump and part of the common family functions for deciding the number of found locally minimal solutions. Other useful features included are the possibility of initializing a population uniformly in a proper range for the used function, an counter for function evaluations, and the ability to acquire the number of minima a function contains.

The software package is freely available in: `http://www.ronkkonen.com/generator/`. The package includes the source codes, a simple plotter program for visualizing the 2D functions generated, written in Matlab, and detailed documentation.

4.4.8 Future work

The framework allows easy addition of new families. One such future family could be a variant of the current cosine family where the global optima would not be located on the constraints. This would allow easy comparison of algorithms in regard to their ability to locate optima on the constraints and offer a different challenge for methods that rely on the constraints and initial population. Another area in which the framework should be developed is the ability to generate functions with a desirable global structure. For the quadratic family this could be achieved by adding an option to give the optima locations as parameters, instead of always generating them randomly. A new family based on the composition idea discussed in section 4.4.3 could also be added to allow the generation of varying landscapes having a large number of optima without excessive computational cost. Additionally, more known test functions could be added to the common family, and some of the existing functions currently limited to two dimensions could be generalized to allow higher dimensional versions.

4.5 Summary

This chapter has considered the topic of evaluating optimization algorithms experimentally, especially in the context of multimodal optimization. Three main challenges for performing quality experimental studies were identified: the selection of suitable performance measures and end condition for the search are important for achieving meaningful results. Also extensive enough parameter studies are required to analyze the potential of different algorithms and the effects of different parameters properly. Most importantly, the selection of the used test function setup is crucial for the success of the study in acquiring useful information about the features of the evaluated optimization algorithms. The features of the test functions should be understood well enough to be able to analyze the reasons behind the good or poor performance of the analyzed algorithms. Some of the most important problem features which can be exploited by different algorithms, were listed and analyzed, as well as some of the most used performance measures.

A software framework for generating multimodal test functions was created and presented in response to the limited selection of test problems available at the moment. The framework consists of four families of parameterizable functions, and more can be added in the future. The families allow comparisons of multimodal optimization algorithms with different and controllable functions having well understood features, allowing the analysis of the algorithms in exploiting the different problem features.

The author's own work of comparing DE and G3-PCX algorithms was used as an example on demonstrating the above mentioned challenges in evaluating optimization algorithms. The results are also used to justify the selection of the DE algorithm as the base for multimodal optimization approaches proposed in the next chapter.

Chapter V

Differential Evolution

Differential Evolution [125, 122, 79] is a population-based steady-state EA introduced by Storn and Price [157]. Several different DE strategies have been suggested [155, 42, 125]. A naming notation is presented in [158] to classify the variants so that each strategy is named as DE/x/y/z, where x defines the target of the mutation operation, y is the number of difference vectors used in the mutation, and z denotes the used crossover scheme. Using the notation, the *basic DE*-strategy is named as DE/rand/1/bin. This states that the mutation target is selected randomly among the population, mutation is performed using a single difference vector, and uniform crossover (crossover due to independent binomial experiments) is used.

The DE strategy used throughout this thesis is the DE/rand/1/bin. The strategy was selected because of its simplicity and popularity, but it has also demonstrated a good performance in difficult multimodal problems [100] and also in the test setup presented in Section 4.3. The algorithm starts from a random initial population. In each generation g, DE creates a trial vector $\vec{u}_{i,g}$ for each target vector $\vec{x}_{i,g}$ of the population using mutation

$$v_{j,i,g} = x_{j,r_0,g} + F \cdot (x_{j,r_1,g} - x_{j,r_2,g}) \tag{5.1}$$

and crossover

$$u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } rand[0,1] \leq CR \vee j = j_{rand} \\ x_{j,i,g} & \text{otherwise} \end{cases} \tag{5.2}$$

operations. Indices r_1, r_2, and r_0 are mutually different and drawn from the set of population indices. $rand[0,1]$ is a random number drawn anew for each i from uniform distribution in the range [0,1]. The difference between two randomly chosen population vectors $(\vec{x}_{r_1,g} - \vec{x}_{r_2,g})$ defines the magnitude and direction of the mutation. This exhibits a self-adaptive behavior to the mutation operation, as the average mutation step length decreases as the population converges. To prevent crossover from duplicating the objective vector, $\vec{u}_{i,g}$ always inherits the parameter with a randomly chosen index j_{rand} from $\vec{v}_{i,g}$. The control parameters for DE are the crossover rate CR, the mutation factor F and the population size NP. $F \in]0, 1+]$ (the "+" sign marks that in principle $F > 1$ can be used, but in practice they are rarely useful) is a scaling factor for the mutation step length and affects the convergence speed of the population. $CR \in [0,1]$ controls the crossover by determining the average proportion of parameters the trial vector $\vec{u}_{i,g}$ inherits from the mutated vector $\vec{v}_{i,g}$.

At the end of each generation the selection operation:

$$\vec{x}_{i,g+1} = \begin{cases} \vec{u}_{i,g} & \text{if } f(\vec{u}_{i,g}) \leq f(\vec{x}_{i,g}) \\ \vec{x}_{i,g} & \text{otherwise} \end{cases} \tag{5.3}$$

is performed comparing each trial vector $\vec{u}_{i,g}$ to the corresponding target vector $\vec{x}_{i,g}$. If the trial has equal or lower cost, it replaces the target vector. DE/rand/1/bin is described in Algorithm 2.

Algorithm 2 DE/rand/1/bin (DEGS)

```
1: Initialize population, $g = 1$
2: while termination criterion not met do
3:   for $i = 1; i \leq NP; i = i + 1$ do
4:     Randomly pick $r_0, r_1, r_2 \in \{1, 2, \ldots, NP\}, r_0 \neq r_1 \neq r_2 \neq i)$
5:     Randomly pick $j_{rand} \in \{1, 2, \ldots, D\}$
6:     for $j = 1; j \leq D; j = j + 1$ do
7:       Perform mutation using (5.1)
8:       Perform crossover using (5.2)
9:     end for
10:  end for
11:  for $i = 1; i \leq NP; i = i + 1$ do
12:    Perform selection using (5.3)
13:  end for
14:  $g = g + 1$
15: end while
```

5.1 Rotational invariance and stagnation

The uniform crossover operation is not rotationally invariant [125, p.101], meaning that the search is biased in the axis directions. This is efficient when optimizing problems with low epistasis, but harmful with nonseparable functions. As separable functions can be solved efficiently by optimizing each parameter independently, EAs are not the most efficient approaches for such problems. The global search capability provided by the usage of population in EAs is better suited for harder nonseparable problems, for which all parameters must be optimized at once. For this reason, rotationally invariant operators are preferred.

The parameter *CR* controls the degree of rotational invariance of DE: the smaller the used value of *CR*, the less rotationally invariant the search becomes. When the crossover is disabled ($CR = 1$), the search becomes completely rotationally invariant. The downside of using *mutation-only* DE is a reduced pool of potential trials [80], which may lead to *stagnation* with small population sizes. Stagnation is a situation in which the population has not fully converged, but the algorithm is not capable of producing new solutions which would improve the fitness. Using value $F = 1$ similarly decreases the pool of potential trials. Increasing the population size, on the other hand, decreases the risk of stagnation.

5.2 Randomized mutation scale factor

Stagnation can be prevented in mutation-only DE by making the F a random variable instead using a fixed value. Zaharie [182] multiplies the mutation differential component-wise with a D dimensional vector $\vec{F}^{jitt}$ of random numbers, drawn anew for each mutation from normal distribution, with expectation 0 and standard deviation σ. Such approaches, where each parameter of the differential vector is multiplied with a different number has been named *jitter* [125, p.80].

In addition to scaling the length, the jitter mutation also changes the direction of the differential. Assuming that none of the variables has identical value for the entire population, which would reduce the corresponding term in the differential to zero, and that the used probability distribution is unbounded, the jitter mutation can theoretically produce any point in the search space with positive probability. Zaharie [182] demonstrates that DE using jitter with unbounded probability distribution fulfills the convergence criteria presented by Rudolph [130] for a theoretically convergent algorithm, i.e. for an algorithm which will converge to the global optimum with probability one in finite time. The proof is based on the fact that the mutation can theoretically produce any point in the search space and DE is elitist, i.e. it never loses the best found solution. While the assumption of having no zero terms in the differential can be safely made in theoretical analysis with infinite precision, in practice such a situation may arise with limited precision.

A downside of using the jitter mutation in replacing the crossover is that jitter is also not rotationally invariant [125, 156]. Another approach to randomize the mutation scale factor is to simply replace the constant F in equation 5.1 with a randomized F^{dith}. Ökdem [188] for example uses uniformly distributed random values in range [-2,2]. Such approaches, where each component of the differential is multiplied by the same random number is called *dither* in [125, p.80]. Dither randomizes the length of the differential, but does not affect its direction, so the mutation can only reach points in a line along the differential vector. This invalidates the convergence proof by Zaharie [182], even if an unbounded probability distribution is used for generating the F^{dith}, as dither mutation is not able to potentially produce any point in the search space. While dither has not been shown convergent in theoretical sense, the method offers a less disruptive way of increasing the pool of potential trials compared to jitter. Because dither only affects the length of the differential, it retains the rotational invariance of the mutation in basic DE. The results of Das et al. [32, 31] suggest that the use of dither is able to increase the performance of DE in certain problems, especially in noisy functions. Storn [156, p. 11] states that “Since dither is rotationally invariant and preserves the contour matching property, this diversity enhancing method should always be used.”.

5.2.1 Study on jitter

The jitter mutation concept has been studied by the author [134, 132]. DE using a similar jitter approach as that of Zaharie [182] was compared to unmodified DE on ten constrained optimization problems taken from [64]. The expectation was set to F and for σ values in range [0,0.1] were tested. Note that using $\sigma = 0$ reduces the method to normal mutation. The constraint handling was done using a method proposed by Lampinen [78, 77].

The results demonstrate the ability of jitter to reduce the number of trials identical to the current population members and to the other trials generated during the same generation. While producing identical trials is clearly redundant in advancing the optimization process, the non-identical solutions may also be of poor quality and thus are not automatically better in advancing the search. The results did not show a significant improvement of performance for the jitter version over the basic DE over the chosen test set. Some interesting findings are worth noting, however, and are presented next.

Figures 5.1 and 5.2 demonstrate the performance of basic DE and the jitter approach as the function of *NP* with a few different parameter settings in a reactor design problem (number 104 in [64, p.113]) with $\varepsilon = 10^{-5}$ and $NFE_{max} = 500000$. The problem is eight-dimensional and has six constraint functions. Figures 5.3 and 5.4 demonstrate the performance on an alkylation process problem (number 114 in [64, p.123]) with $\varepsilon = 1$ and $NFE_{max} = 300000000$. The problem is ten-dimensional and has 11 constraint functions, three of which are equality constraints. The equality constraints were substituted with two inequality constraints, leaving a range of 0.002 between them and increasing the actual number of constraint functions to 14.

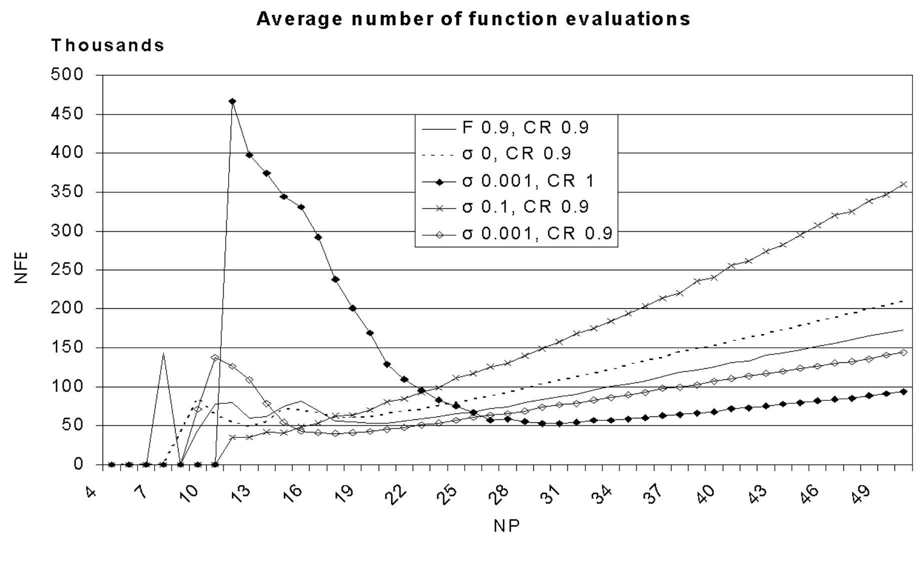


Figure 5.1: Average number of function evaluation from 1000 independent runs for problem number 104 in [64, p.113].

For both problems, the expected effect of the variation operators (mutation and crossover)

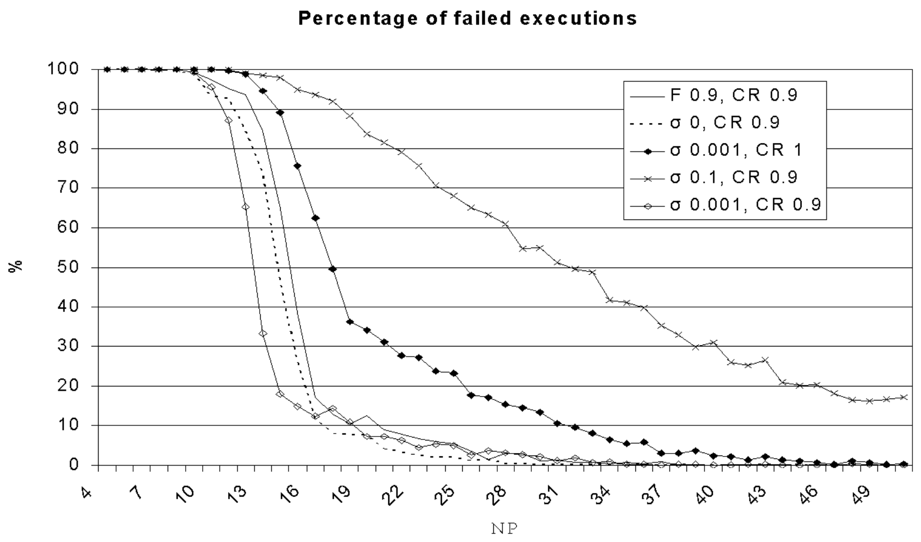


Figure 5.2: Percentage of runs which failed to locate the global optimum with precision 10^{-5} within 500000 function evaluations for problem number 104 in [64, p.113].

of DE to the variance of population c^{var} was kept constant by using a dynamic value for expectation F for generating the $\vec{F}^{jitt}$ calculated by

$$F = \sqrt{\frac{c^{var2} - 1}{2CR} + \frac{2 - CR}{2NP}} \tag{5.4}$$

The equation follows from solving the F from the definition of c^{var} by Zaharie [182]:

$$c^{var} = \sqrt{2F^2CR - 2CR/NP + CR^2/NP + 1} \tag{5.5}$$

With $c^{var} < 1$, the variation operators decrease, and with $c^{var} > 1$, the variation operators increase the population variance. As the variation operators are expected to increase the population variance in contrast to the selection operation, values $c^{var} < 1$ would lead to a premature convergence as none of the operations would be able to increase the population diversity. Zaharie [182] suggest a usable range for the c^{var} parameter to be [1,1.5]. To emphasize robustness, the value $c^{var} = 1.5$ was used in equation 5.4 for calculating the F.

The results suggest that the value for σ in generating the $\vec{F}^{jitt}$ for equation 5.9 should be rather small, somewhere in the range [0, 0.1]. As can be seen in Figures 5.1 and 5.2, the performance of DE is clearly inferior with $\sigma = 0.1$ compared to the smaller values. Using large values for σ makes the mutation behave like random search, as the differentials have only little meaning and the algorithm loses its ability to use the population information

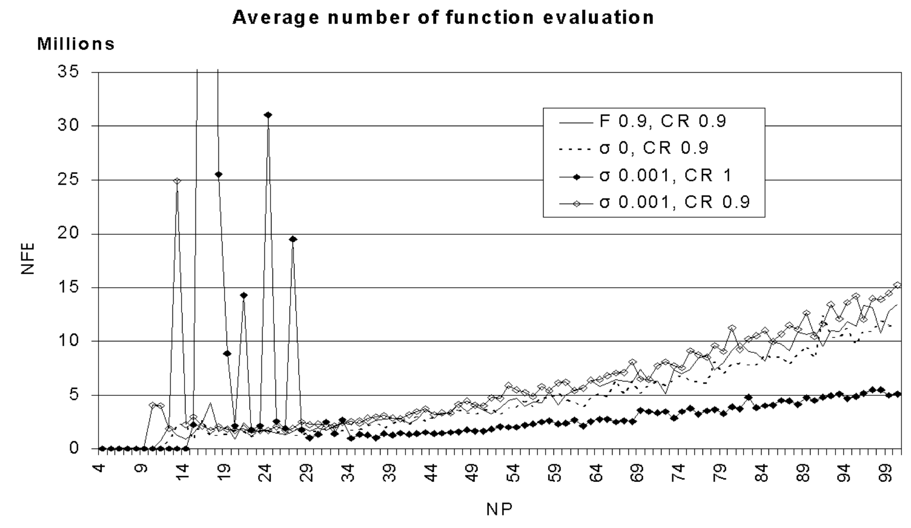


Figure 5.3: Average number of function evaluations from 10 independent runs for problem number 114 in [64, p.123].

for its advantage. Additionally, the lack of rotational invariance of jitter affects the search more with increasing values of σ.

Figures 5.1 and 5.2 demonstrate the typical effect of population size on the performance of DE. With a too small population size, the algorithm will often converge prematurely and fail to locate the global optimum. Typically an optimal *NP* exist, which is enough to provide reliable convergence with a minimal number of function evaluations. If the NP is increased further, the required number of function evaluations increases as well. It is notable that DE requires a larger population size with $CR = 1$ compared to the $CR = 0.9$ to achieve reliable convergence. However, also the required number of function evaluations increases slower along with the increasing population size. A similar effect is visible in Figure 5.3. This demonstrates the advantage of using $CR = 1$ on problems with high epistasis, as the algorithm becomes less sensitive to the selection of the *NP* parameter.

The alkylation process problem was clearly the most demanding among the used test setup. The equality constraints posed a special problem for the used approach, as the inequality constraints used to emulate them form a very narrow area for acceptable solutions, which may be hard to locate. The alkylation process problem was also the only one among the test setup which demonstrated a clear performance difference between the basic DE and the jitter version. As can be seen in Figure 5.4, both setups of the basic DE generate failures with the whole tested population range while the jitter version only fails with $NP < 30$. Because of time limitations, the tests were done using only 10

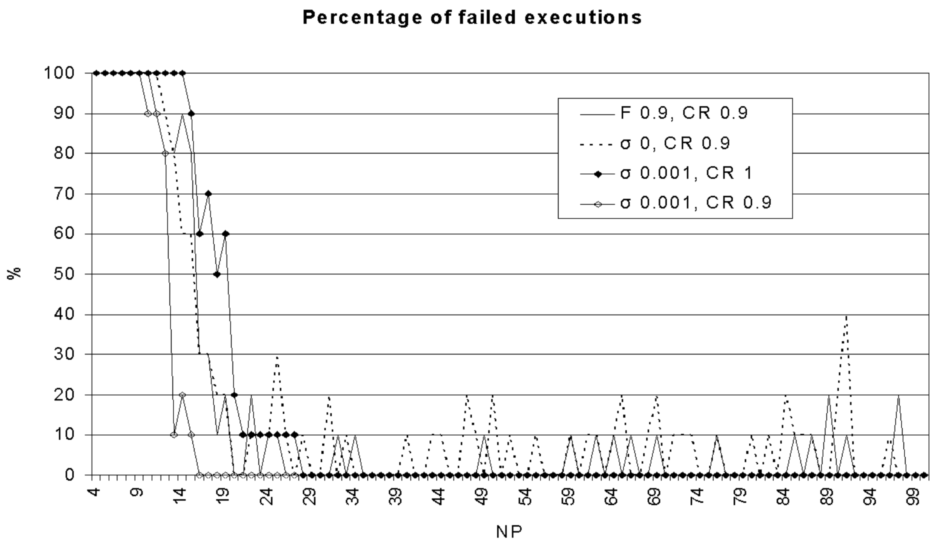


Figure 5.4: Percentage of runs which failed to locate the global optimum with precision 1 within 300000000 function evaluations for problem number 114 in [64, p.123].

repetitions, and also the maximum *NP* of 100 was rather low for such a problem. Thus no definite conclusions can be drawn from the single result. Nevertheless, it suggests that using a randomized mutation scale factor may be beneficial, especially in harder problems.

5.2.2 Additive jitter

To achieve a mutation operator capable of reaching any point in the search space with a positive probability similar to jitter but also capable of retaining the rotational invariance of mutation, an *additive jitter* (ajitter) concept was proposed by the author in [135]. Instead of using the random term in the multiplication, the method adopts the idea proposed by Ter Braak [161, 162] of using addition in implementing the jitter. Ter Braak draws each component of a vector $\vec{F}^{ajit}$ of length D from a symmetric unbounded distribution with expectation 0, using a small fixed value for the standard deviation, and adds the resulting vector to the scaled differential. Using a fixed value for the standard deviation is problematic because it does not scale according to the length of the differential. Thus the proportional effect of the added component is dependent on the length of the differential: for long differentials the added component may easily become indistinguishably small, while for short differentials the added component may dominate the mutation. This can disrupt the self-adaptive nature of DE, which relies on changes on population differentials.

In jitter and dither, the effect of randomization is automatically proportional to the length of the differential because of the multiplication. To achieve the same with the ajitter, the standard deviation must be scaled. The vector $\vec{F}^{ajit}$ has thus been constructed by drawing random numbers from a normal distribution with expectation 0 and using the length of the differential scaled by $F\sigma/\sqrt{D}$ as standard deviation:

$$F_j^{ajit} = N\left(0, \|\vec{x}_{r_1,g} - \vec{x}_{r_2,g}\| \cdot F\sigma/\sqrt{D}\right), j = 1, 2, \ldots, D \tag{5.6}$$

The scaling term $\sqrt{D}$ keeps the length of $\vec{F}^{ajit}$ independent of the dimension of the problem. σ represents the proportional length of the differential. The use of σ here in a slightly different meaning compared to jitter and dither, where it directly defines the standard deviation of the used normal distribution, is intentional. The effect of changing the value of the σ parameter has a comparable effect in all three approaches, and the σ becomes an added parameter for a DE using any of the methods.

The convergence proof of Zaharie [182] applies to the ajitter in a similar manner as to the jitter. Additionally, the ajitter is also a rotationally invariant process because the construction of $\vec{F}^{ajit}$ uses a symmetric distribution. Figure 5.5 demonstrates this by displaying the distributions acquired by generating 400 random points for three differently aligned differentials of equal length using dither, jitter and ajitter mutation with $\sigma = 0.1$ and $F = 1$. As can be seen, the shapes of the distributions generated by dither and ajitter are not affected by the rotation of the differential. For jitter, however, the shape of the distribution clearly changes along the rotation.

5.3 Selection pressure

The selection operation of DE/rand/1/bin is *global* in the sense that the base vector for mutation is a randomly selected population vector other than the target vector. This means that the trial vector $\vec{u}_{i,g}$ has no relation to the target vector $\vec{x}_{i,g}$, against which it competes in the selection phase (eq. 5.3). Global selection thus allows underperforming population vectors to be replaced with variants of the better solutions of the population. Most of the traditional DE variants use global selection. The abbreviation *DEGS* stands for DE using global selection and is used in this thesis as a synonym for the DE/rand/1/bin, as it is the only traditional DE variant used.

In *local selection*, equation 5.1 is modified so that the target and base vectors are the same ($r_0 = i$). Now each population member is always compared to its own mutant in the selection phase. When the selection is local, evolution of a vector only depends on the current set of vector differences, and not directly on the parameter values of vectors other than those of the target. In effect, local selection partitions the population into *NP* niches, each of which is inhabited by a single vector that evolves in isolation. The abbreviation *DELS* is used for DE using local selection.

Ter Braak [161, 162] demonstrates that local selection is Markovian and satisfies the detailed balance condition [127, p. 229] and transforms a mutation-only DELS to a Markov chain Monte Carlo (MCMC) algorithm by introducing a probabilistic Metropolis-Hastings acceptance rule. MCMC algorithms aim to generate an output for matching a target distribution. The resulting method, *Differential Evolution Markov Chain*, runs

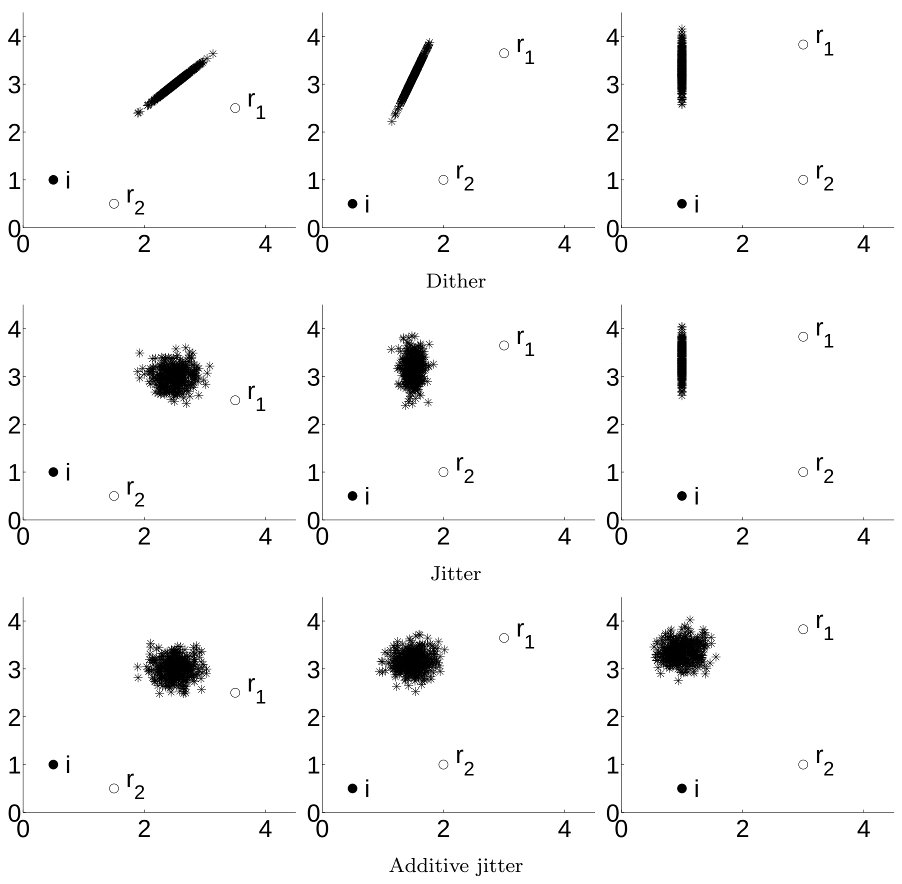


Figure 5.5: Graphical presentation of 400 random points produced by dither (top row), jitter (middle row) and additive jitter (bottom row) mutation operations with $\sigma = 0.1$ and $F = 1$. The points i, r_1 and r_2 represent the end points of vectors $\vec{x}_i$, $\vec{x}_{r_1}$ and $\vec{x}_{r_2}$ starting from point $0, 0$ and the $r_1 - r_2$ is the differential which is added to i.

NP parallel Markov chains. DE provides the scale and orientation for the proposal distribution. Based on an analogy with the classic random walk Metropolis algorithm, the optimal choice for the mutation scale factor in such a setup is $F^* = 2.38/\sqrt{D}$ [128].

The *drift-free DE* proposed by Price [123] uses the local selection in combination with a drift-free recombination operation. The approach aims to eliminate biases caused by crossover, global selection and the *three-vector arithmetic recombination*, which is suggested in [125, p. 108] as a replacement for the crossover. In drift-free DE, the mutation is responsible for exploring the search space, while recombination homogenizes the population.

5.3.1 Scaling study

In order to study the differences between DEGS and DELS, a scaling study was performed by the author in [124]. Especially the effect of dimensionality on the optimal values of parameters *NP* and *F* and their mutual interactions were studied using simple convex unimodal functions. Mutation-only versions ($CR = 1$) of both algorithms were used to eliminate disruptions caused by the crossover and, reduce the number of tunable control parameters.

The test setup

A set of experiments was performed using three convex unimodal functions with quadratic minima: a sphere (eq. 4.5), an ellipse (eq. 4.6), and a rotated ellipse (eq. 4.7). Highly multimodal functions were not considered because the aim was to observe fundamental differences between the two forms of selection without the complicating influences entailed by functions whose properties change with dimensionality. However, the generalized Rosenbrock function (eq. 4.8) was included as the final function of the test bed to test the effect of the non-convexity on the performance of the methods.

Initially the average NFE from 100 independent runs was calculated for each combination of $D = [2, 4, \ldots, 30]$, $F = [0.1, 0.2, \ldots, 1]$ and $NP = [1D, 2D, \ldots, 10D]$. For the convex functions, the initialization of population was done using uniformly distributed random numbers in the range [-100, 100] and in the range [-30, 30] for f_{ros}. Additionally, $D = 50$ was tested by using 10 independent runs. For DEGS, the *NP* range was extended up to $14D$ for the convex functions and up to $16D$ for the f_{ros}. The performance was measured by calculating SP using $\varepsilon = 10^{-6}$ and $NFE_{max} = 1000D \cdot NP$. The resulting graphs are available in `http://www.it.lut.fi/ip/evo/`.

To provide more accurate estimates for the SP^* (the optimal SP), the F^* and NP^* (the values of *NP* and *F* which produced the SP^*) for the convex functions, the *F* and *NP* were sampled at higher resolution centered on the estimates of their optimal values from the initial setup and averaged over 400 runs. In particular *F* was sampled in the range $[F^* - 0.10, F^* - 0.08, \ldots, F^* + 0.10]$ and *NP* from $NP^* - D/2$, NP^* and $NP^* + D/2$.

results

Figure 5.6 plots the achieved SP^*, NP^* and F^* as a function of dimension. The trend of each is estimated by displaying the best-fit power law trend lines. The results for all three convex functions are practically identical, demonstrating that neither the disparate magnitudes of parameters nor the rotation have a significant effect on the performance of either version of mutation-only DE, as can be observed in Figures 5.6(a), 5.6(c) and 5.6(e). This is expected, as the mutation operation is rotationally invariant, and the use of differentials is efficient for scaling the step length of the mutation.

Figure 5.6(a) demonstrates that while DEGS is initially faster on the convex functions, DELS gives a lower SP^* once $D > 14$. A best-fit power law trend line for the DELS data reveals that SP^* nearly increases in proportion to D^2, whereas the corresponding best-fit power law trend line for DEGS shows SP^* increasing faster than $D^{2.5}$.

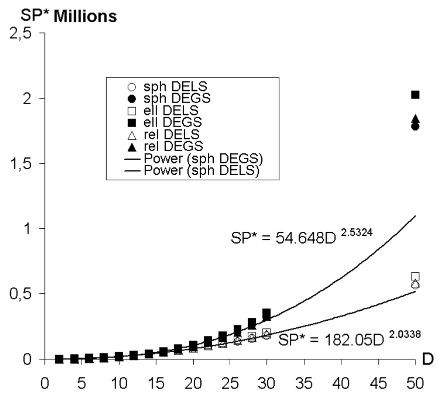
As the trend line in Figure 5.6(c) shows, the optimal population size for DELS increases linearly with dimension in each convex problem corresponding to $NP^* = 2D$, except for $D = 2$ when $NP^* = 5$. This relationship between NP^* and D remains valid even at $D = 50$. Given that sphere is such an easy function to optimize, the $NP^* = 2D$ would appear to predict the smallest possible reliable population size for any function for the mutation-only DE. Choosing a larger than optimal population size, for example $NP = 4D$, remains effective and makes DELS less sensitive to the choice of F. The corresponding plot for DEGS shows that NP^* roughly increases in proportion to $D^{1.4}$.

While the F^* for DEGS in Figure 5.6(e) seems to be approaching a lower limit at around $F = 0.5$, the optimal scaling constant for DELS continues to decrease as D increases. The power law trend line reveals that under DELS, F^* very nearly decreases in inverse proportion to the square root of D.

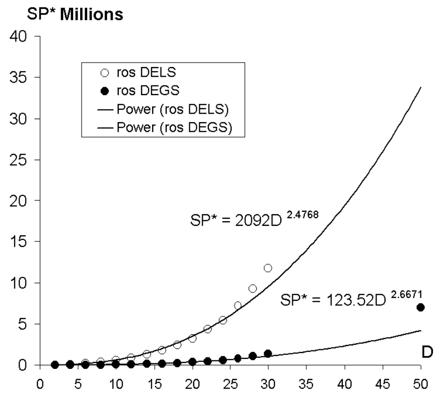
Comparing the results for the Rosenbrock function in Figures 5.6(b), 5.6(d) and 5.6(f) to the corresponding ones for convex functions shows the effect of the banana-shaped contours on DELS and DEGS. Even though the simple power law is no longer a good fit, Figure 5.6(b) suggests that SP^* increases faster for both methods, compared to the three convex functions. For DEGS, the exponent grows from 2.5 to approximately 2.7, and the constant factor about doubles when moving from convex functions to the Rosenbrock function. While the exponent of the trend line for DELS is roughly 2.5, a little less than for DEGS, the almost twenty times bigger constant factor makes the algorithm clearly slower in terms of SP. The SR for DELS drops along with increase in the number of dimensions, and in 50-dimensional Rosenbrock the algorithm fails to solve the problem within the given NFE_{max}.

Figure 5.6(d) shows that over the range $D = [2, 30]$ the NP^* scales almost linearly in proportion to D for both algorithms, although the lone DEGS result at $D = 50$ suggests that at some point NP^* begins to increase faster. DELS requires slightly larger populations compared to DEGS on this function. Thus while non-convexity left the scaling performance of NP unaffected except for the increased constant factor, it actually lowered somewhat the rate of growth for DEGS. However, because DELS was unable of solving the Rosenbrock with $D = 50$, the result is inconclusive.

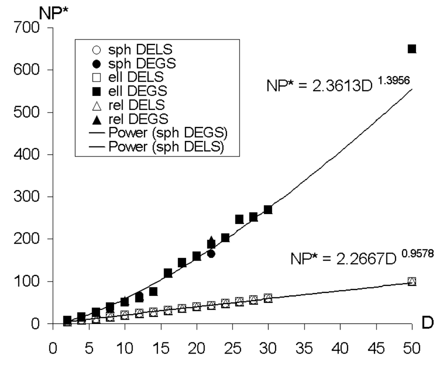
The limit of 100 runs conducted for the Rosenbrock function makes its scaling data for F^* less reliable. The two plots in Figure 5.6(f) share similar profiles, except that the plot for DELS is shifted down using lower F^* and to the right (higher D) compared to the plot for DEGS. For both algorithms, F^* exhibits an initial rise followed by a slow decline to what appears to be a constant value. For DELS, F^* rises to 0.4 when $D = 10$, before dropping to 0.2 for $D > 20$. Similarly, F^* for DEGS rises to 0.8 at $D = 4$ before



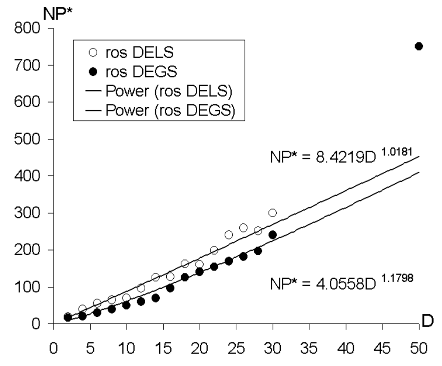
(a) SP^* for convex functions



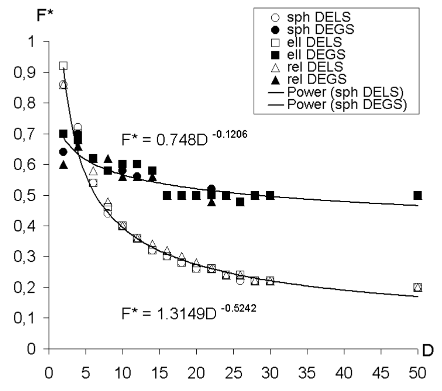
(b) SP^* for the Rosenbrock function



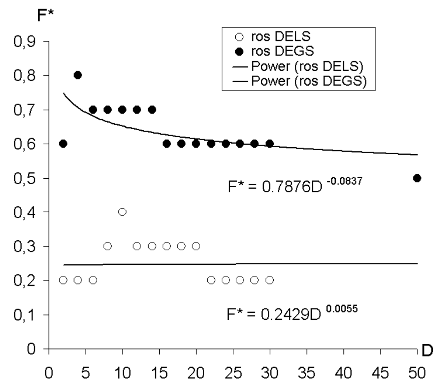
(c) NP^* for convex functions



(d) NP^* for the Rosenbrock function



(e) F^* for convex functions



(f) F^* for the Rosenbrock function

Figure 5.6: The SP^*, NP^* and F^* for DEGS and DELS as a function of dimension and corresponding best-fit power law curve. For the convex functions, the trend line is only shown for the sphere function.

approaching $F^* = 0.5$ as a limit. The non-convexity appears to lessen the disparity in the scaling performance that DELS and DEGS exhibited on convex functions, and demonstrates that the results derived from convex quadratic functions are not universal.

Discussion and conclusions

A best-fit power law curve for the DELS data on the convex unimodal functions demonstrates that F^* is roughly equal to $1.3/\sqrt{D}$. This result resembles the theoretically predicted value of $F^* = 2.38/\sqrt{D}$ for the random walk Metropolis algorithm when the target distribution is Gaussian in the limit of large D [128]. It is also similar to the pre-factor of $1/\sqrt{2D}$, with which the Evolution Strategies scale a multivariate normal distribution [19]. For DEGS, however, F^* never drops below 0.48, because decreasing F inflates the optimal population size. Why then do small scale factors entail large populations for DEGS but not for DELS?

To understand the performance difference between DELS and DEGS, it helps to consider what happens when F approaches zero. When the selection is local, the trial vector will resemble the target vector $\vec{x}_{i,g}$ more closely when F nears zero. Under global selection, the shrinking F makes the trial vector more like a random base vector $\vec{x}_{r_0,g}$. When $F = 0$, DELS compares each vector to itself, while DEGS compares each vector to a randomly chosen population vector. When DELS replaces a vector by its copy (as it always does because the copy has the same objective function value as the vector it is replacing), there is no change in the composition of the population. DEGS, however, changes the composition of the population when it replaces the target vector with a different vector of an equal or lower objective function value. While not every base vector will be better than the target vector that it tries to replace, those that are accepted can quickly drive the population to uniformity.

Another way to explain this behavior is that short takeover time of global selection is responsible for limiting the effectiveness of small values of F. Goldberg and Deb [54] define the takeover time of a selection method as the expected number of generations until copies of the best vector fill an initially diverse population, when selection is the only used operation (no variation operations like mutation or recombination are used). In principle DEGS could choose the best vector as a base vector NP times in a single generation. In this unlikely case, the minimum takeover time for global selection is a single generation, i.e. after NP competitions, the best vector would have filled the whole population with copies of itself. Of course, the average takeover time of global selection will be much longer because it is very unlikely that the same vector will be randomly chosen as a base vector NP times in succession. Through a simulation, the takeover time of DEGS has been estimated as proportional to $NP \cdot log(NP)$ in a limit of large NP [124]. Either way, the takeover time of DEGS depends on NP. By contrast, the takeover time for DELS is infinite, because local selection operating alone can never propagate the best solution out of its own niche. Consequently, NP does not have to be inflated to extend the takeover time long enough for the population to find a solution before becoming uniform.

The ability of local selection to de-couple the population size from its strong dependence on the mutation scale factor allows small populations to be used in conjunction with small F. Under DEGS, the population sizes must be expanded beyond what DELS requires in order to prevent premature convergence. Once the optimal scale factor for the problem at hand drops below $F = 0.5$, the increase in population size begins to degrade the SP for DEGS, whereas DELS remains both reliable and efficient. However, the results for the Rosenbrock function demonstrate that the ability to use a small scale factor is not always helpful. Especially on multimodal functions, small values for F become problematic, because the algorithm loses the ability to use the population information effectively to identify and exploit the global features of the problem. Instead, using small F with DELS basically incorporates an explicit PN method for the DE, as each population member is actually doing extended local search within the area defined by the scaled mutation differentials.

5.3.2 Extending DELS for multimodal problems

To employ the potential of DELS to benefit from the use of small F for multimodal problems, the mutation operation was divided into two parts by the author in [135]: *local mutation* and *global mutation*. The resulting algorithm, DELS using local mutation (DELL), adopts the "either/or concept" [125, p.117], which uses only one variation operator for generating each trial. The selection between two possible operators is done probabilistically for each trial using the *PX* parameter to control the probabilities for using either. While it would be possible to use the traditional uniform crossover operation also with DELS, it would destroy the rotational invariance of the approach, and thus the crossover has been removed from the proposed algorithm.

The local mutation operator

$$\vec{u}_{i,g} = \vec{x}_{i,g} + 1.3/\sqrt{D} \cdot (\vec{x}_{r_1,g} - \vec{x}_{r_2,g}) \tag{5.7}$$

is responsible for offering an efficient local search capability to the algorithm. It is simply the normal DELS mutation using the optimal mutation scale factor $1.3/\sqrt{D}$ for convex-unimodal problems (Figure 5.6(e)).

The global mutation is responsible for the global phase of the algorithm and allows solutions to escape local optima. The idea is to use a similar mutation operator as in the local selection, but using a longer mutation step length. Value $F = 1$ is preferable, as it means that the mutation step length is the unscaled length of the differential and represent the actual scale of the population. Additionally, the dither, jitter or ajitter concepts can be used in the global mutation to increase the pool of potential trials. The dither version of the global mutation uses a random number F^{dith}, drawn anew for each mutation from normal distribution, with expectation F and standard deviation σ

$$\vec{u}_{i,g} = \vec{x}_{i,g} + F^{dith} \cdot (\vec{x}_{r_1,g} - \vec{x}_{r_2,g}) \tag{5.8}$$

in scaling the differential. The abbreviation *DITH* is used for DELL using the dither global mutation. In the jitter version, the differential is multiplied component-wise with a D dimensional vector F^{jitt} of random numbers, drawn anew for each mutation from normal distribution, with expectation F and standard deviation σ

$$\vec{u}_{i,g} = \vec{x}_{i,g} + \vec{F}^{jitt} \otimes (\vec{x}_{r_1,g} - \vec{x}_{r_2,g}) \tag{5.9}$$

The abbreviation *JITT* is used for DELL using the jitter global mutation. The ajitter version of the global mutation adds the $\vec{F}^{ajit}$ to the differential

$$\vec{u}_{i,g} = \vec{x}_{i,g} + (\vec{x}_{r_1,g} - \vec{x}_{r_2,g}) + \vec{F}^{ajit} \tag{5.10}$$

New $\vec{F}^{ajit}$ is constructed for each mutation operation according to Equation 5.6. The abbreviation *AJIT* is used for DELL using the ajitter global mutation.

DELL is described in Algorithm 3. Compared to the basic DE, two new parameters are required, *PX* and σ. On the other hand, the parameter *CR* is no longer required, as the crossover is removed. Thus the actual number of parameters increases by one compared to the basic DE. For F, the value of 1 is recommended to keep the global mutation really global. However, F has been left here as a parameter because it will be demonstrated later in this thesis that using a different value may be beneficial in some cases. The frequency to use either local or global mutation is controlled by PX, so that with $PX = 0$ global mutation is never used and with $PX = 1$ global mutation is always used. σ controls how large role the differential has in the global mutation. Setting $\sigma = 0$ reduces all three versions to basic DELL, where the differential solely defines the mutation, and increasing the value takes the global mutation towards a random search.

Algorithm 3 DELL, the initial version

```
1: Initialize population, $g = 1$
2: while termination criterion not met do
3:   for $i = 1; i \leq NP; i = i + 1$ do
4:     Randomly pick $r_1, r_2 \in \{1, 2, \ldots, NP\}, r_1 \neq r_2)$
5:     if $rand[0,1] < PX$ then
6:       Perform global mutation using (5.8), (5.9) or (5.10)
7:     else
8:       Perform local mutation using (5.7)
9:     end if
10:  end for
11:  for $i = 1; i \leq NP; i = i + 1$ do
12:    Perform selection using (5.3)
13:  end for
14:  $g = g + 1$
15: end while
```

Test setup

Five test functions, Rosenbrock (eq. 4.8), Rastrigin (eq. 4.8), Schwefel (eq. 4.10), Whitley (eq. 4.13) and Rana (eq. 4.16), were used to test the suggested three variants of DELL, and for comparison, similar tests were performed with DEGS. For each function, a scaling study for different dimensions was performed. Two different initialization ranges were tested for f_{ras} to see if the placement of an optimum has an effect on the performance of the algorithms. Randomly rotated versions of Rastrigin (f_{rras}) and Schwefel (f_{rsch}) were also included to change the functions to a nonseparable form. The used rotation method is similar to the one used in Section 4.4.2 for rotating the cosine family functions.

The final included variant of the Rastrigin function is a randomly rotated and miss-scaled version:

$$f_{mras}(\vec{y}) = 10D + \sum_{i=1}^{D} \left((10^{\frac{i-1}{D-1}} y_i)^2 - 10\cos(2\pi \cdot 10^{\frac{i-1}{D-1}} y_i) \right). \tag{5.11}$$

Table 5.1 summarizes the used test function setup.

Table 5.1: The settings for different problems.

Function	Dimension	NFE_{max}	Used constraints
f_{ros}	2-50	$1000D \cdot NP$	$-30 \le x_j \le 30 \quad j = 1, \ldots, D$
f_{ras}	2-20	$3000D \cdot NP$	$-5.12 \le x_j \le 5.12 \quad j = 1, \ldots, D$ and $-0.12 \le x_j \le 10.12 \quad j = 1, \ldots, D$
$f_{rras,mras}$	2-20	$3000D \cdot NP$	$-5.12 \le y_j \le 5.12 \quad j = 1, \ldots, D$
f_{sch}	2-20	$3000D \cdot NP$	$-512 \le x_j \le 512 \quad j = 1, \ldots, D$
f_{rsch}	2-20	$3000D \cdot NP$	$-512 \le y_j \le 512 \quad j = 1, \ldots, D$
f_{whi}	2-20	$3000D \cdot NP$	$-5.12 \le x_j \le 5.12 \quad j = 1, \ldots, D$
f_{rana}	2-5	$20000D \cdot NP$	$-520 \le x_j \le 520 \quad j = 1, \ldots, D$

The performance was measured by calculating SP from 100 independent runs. For f_{ros} $\varepsilon = 10^{-6}$ and for the other functions, $\varepsilon = 10^{-2}$ was used. However, when SR is very small, the value of SP is not very informative, as a small change in SR results in a huge change in SP, and the variance in the SR becomes an issue. For this reason, the SP values were calculated only for cases where the problem was considered *solved*. In this study, a value of $SR > 5/100$ was required to categorize the problem as solved. A term *occasional success* is used to refer to the runs for which $SR \le 5/100$. The full set of results is available in `http://www.it.lut.fi/ip/evo/`.

The constraints were handled by replacing the faulty variable with a randomly selected value inside the constrained area. To save computation time, higher dimensional runs were not performed, if the algorithm had already failed to solve a lower dimensional version of the problem with all tested parameter combinations.

Because the test setup was very time consuming, not all possible parameter combinations were tested. For DEGS, the crossover was disabled, using always $CR = 1$ to achieve rotation invariance. For DEGS the values $0.5, 0.6 \ldots 1$ for F were tested in all cases. The values $F < 0.5$ were left out, due to the previous findings, which suggested that DEGS can not fully benefit from small values of F (Figures 5.6(e),5.6(f)). For the DELL methods, $F = 1$ was always used.

The value of $NP = 9D$ was tested with all methods and functions because it was close the optimum NP value identified for the Rosenbrock function (Figure 5.6(d)). Additionally, for DEGS, the value of $NP = 2.4D^{1.5}$, and for the DELL methods, the value of $NP = 3D$ were tested on all problems. These values are close to the optimum NP values identified for convex unimodal problems (Figure 5.6(c)) and can be considered the lower bound for viable NP values for any harder problems. Because DEGS typically requires larger population sizes than DELS, due to the larger selection pressure, the value of $NP = 20D$ was also tested with DEGS in all other problems except f_{ros}. Lastly, with f_{rana} all methods were tested also with values $NP = 20D$ and $NP = 40D$.

For the DELL methods, values of $\sigma = 0.1$, $\sigma = 0.01$ and $\sigma = 0$ were tested for all problems. Additionally, the value of $\sigma = 0.5$ was occasionally used, to get an idea of how much greater σ values would perform, and to make certain that the used magnitude of σ was not too small altogether. These tests were not used, however, when determining the best SP values, because a complete set of results were not generated using $\sigma = 0.5$. The PX value was scaled from $0, 0.2, \ldots, 1$ for all DELL methods and all problems.

Results

All the tested algorithms were able solve the Rosenbrock function regardless of the dimension. As can be seen in Figure 5.7 all DELL methods perform almost similarly and are clearly faster than DEGS. The relative difference between DELL and DEGS grows larger as the dimension of the problem increases.

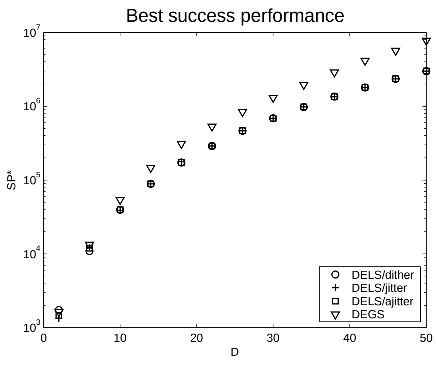
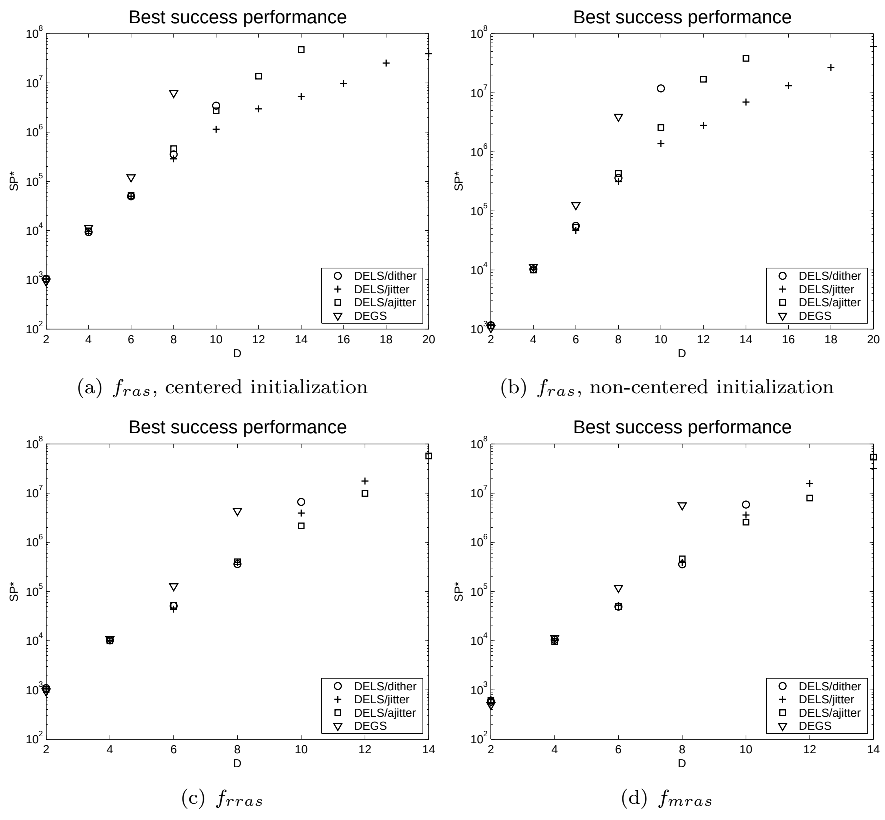


Figure 5.7: Best achieved SP values for f_{ros}

The results for unrotated Rastrigin are presented in Figures 5.8(a) and 5.8(b). Changing the initialization range of population from centered to non-centered does not have a considerable effect on any of the methods. DEGS is able to solve (attain $SR > 5$) when $D \leq 8$. Among the DELL methods, DITH is able to solve the problem when $D \leq 10$, AJIT when $D \leq 14$ and JITT with all tested dimensions. For the SP, DEGS is again the slowest, while JITT is able to outperform DITH and AJIT when $D > 8$.

The situation changes, when the Rastrigin function is rotated, which makes the function nonseparable. As can be seen in Figures 5.8(c) and 5.8(d), the miss-scaling does not have a significant effect on the performance of any of the algorithms. Also the rotation has a negligible effect on the performance of DEGS, DITH and AJIT. However, the results of JITT decrease notably and are now comparable to those of AJIT.

DELL outperforms DEGS also in the Schwefel function, as can be seen in Figure 5.9. DEGS is able to solve the problem when $D \leq 14$ and the rotation again has only a small effect on the performance. AJIT has problems with the higher dimensional versions and is unable to solve (other than occasional successes) the unrotated version with $D = 20$ and the rotated version with $D > 16$. DITH and JITT are able to solve the problem with all tested dimensions. DEGS was the slowest, when comparing SP. Again JITT



(a) f_{ras}, centered initialization

(b) f_{ras}, non-centered initialization

(c) f_{rras}

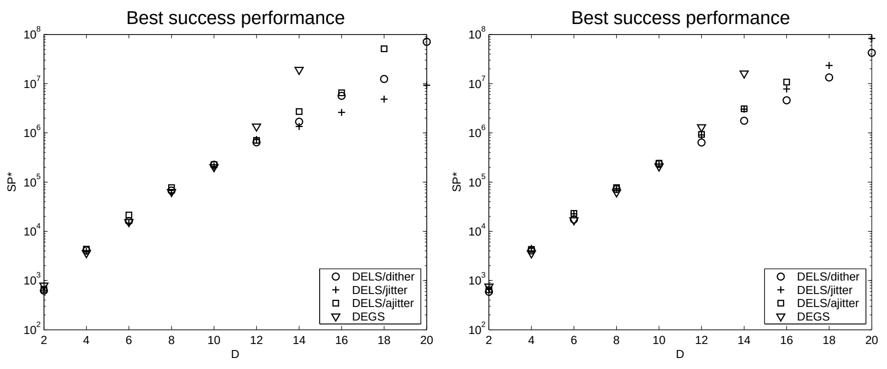
(d) f_{mras}

Figure 5.8: Best achieved SP values for different versions of the Rastrigin function

outperformed the other DELL algorithms in the unrotated version, but in the rotated version DITH was the fastest.

The Whitley function was the only one in the test set where DEGS was able to outperform DELL. DEGS managed to solve the problem when $D \leq 18$, when the DELL methods only managed to solve it when $D \leq 16$ (all methods scored occasional successes with all tested dimensions). When comparing the SP values in Figure 5.10(a), DEGS is clearly faster than DELL, regardless of the dimension. The differences between the performance of the different DELL methods are rather small.

The Rana function (Figure 5.10(b)) was clearly the hardest on the test setup. DEGS was able to solve it only when $D = 2, 3$ while all DELL methods were able to solve it when $D = 2, 3, 4$. DITH was the only method to get occasional successes when $D = 5$. In terms of SP, DEGS was the slowest when $D = 2$ and shared the slowest performance with AJIT when $D = 3$. Among the DELL methods, AJIT was the slowest and JITT the fastest.



(a) f_{sch} (b) f_{rsch}

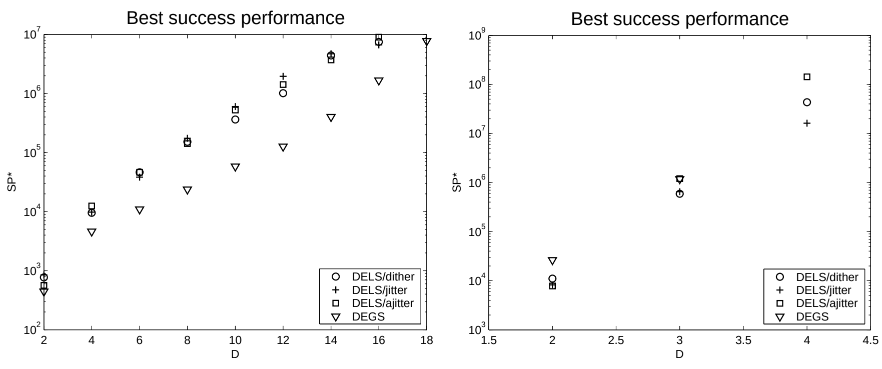
Figure 5.9: Best achieved SP values for the Schwefel function

Effects of control parameters

The value of $PX = 0$ (global mutation disabled) performed poorly in all the tested cases throughout the test set. In Rosenbrock, all DELL methods behaved rather similarly, and small values performed best. Increasing the value up to 0.6 did not have a significant effect in most cases, but using higher values started to decrease the performance of the algorithms. In the other functions, PX did not have a clear optimum value. For DITH, PX^* typically got smaller as the dimensionality of the problem increased. For JITT and AJIT the trend was not as clear.

In most cases, smaller values for σ increased the speed, but decreased the SR. In the Rosenbrock function, small σ values performed generally well, and it was the only tested function where the value of $\sigma = 0$ outperformed the larger values. In Rastrigin and Schwefel, the value of $\sigma = 0.01$ performed well with small D, but with increasing dimensionality, $\sigma = 0.1$ gained the upper hand (except for AJIT in Schwefel) because of increased SR. In the Whitley function, DITH performed well with $\sigma = 0.1$, but for JITT and AJIT the best value for σ was more often 0.01 than 0.1.

The best performing population sizes for all DELS methods on Rosenbrock were $NP = 3D$ for all dimensions, and for DEGS they followed $NP = 2.4D^{1.5}$, as expected. For Rastrigin, all DELL methods performed best with $NP = 9D$, which was also the best population size for DEGS when $D \leq 4$. With the increased dimension DEGS, required $NP = 20D$. In Schwefel, all DELL methods performed best with $NP = 3D$ until $D \leq 8$ and after that with $NP = 9D$. DEGS used $NP = 2.4D^{1.5}$ or $NP = 9D$ when $D \leq 12$ and $NP = 20D$ with $D = 14$. For Whitley, the best population size for DEGS was mostly $NP = 9D$, except for $NP = 2.4D^{1.5}$ with $D = 6$ and $NP = 20D$ with $D = 18$. The DELL methods used $NP = 3D$ in low dimensional and $NP = 9D$ in higher dimensional cases. DITH swapped at $D = 6$, JITT at $D = 8$ and AJIT at $D = 12$. In Rana, DEGS performed best with $NP = 40D$, while the DELL methods used $NP = 3D$ for $D = 2$, $NP = 20D$, for $D = 3$ (except for JITT, which performed best with $NP = 9D$) and $NP = 40D$ with higher dimensions.



(a) f_{whi} (b) f_{rana}

Figure 5.10: Best achieved SP values for the Whitley and Rana functions

Discussion and conclusions

As only a limited number of different parameter combinations were studied for each method, it can not be exclusively claimed that some of the tested methods outperform others even inside the selected function testbed, because with different parameter setup the results may change. Especially, enabling the crossover for unrotated Rastrigin and Schwefel would definitely increase the performance of DEGS, because it could exploit the separability of the functions, as can be seen in the results presented in Section 4.3. The goal of this study was, however, to demonstrate that with a suitable migration operator, the concept of DELS can be applied to general multimodal problems, and that goal was achieved. The performance obtained with DELL compares favorably with the performance of DEGS. Especially in the Rosenbrock function, all DELL methods were able to outperform DEGS. When the results are compared to the ones presented in Section 5.3.1, it can be clearly seen that the use of global mutation operation increases the performance of DELS with the Rosenbrock function significantly.

When comparing the overall results, DELL was able to outperform DEGS in all but the Whitley function. It seems that the combination of a very steep overall slope and a highly multimodal area around the optimum favor the greedier global selection, which draws the population fast to the general area of the global optimum and can then perform efficient local search. It can be observed in the results with different versions of Rastrigin that none of the tested algorithms are sensitive to miss-scaling or moving the optimum away from the center of the search space. Also the results for Rastrigin and Schwefel functions show that rotation has only a small effect on the performance of DEGS, DITH and AJIT, which is consistent with the fact that these methods are rotationally invariant. JITT, on the other hand, performs clearly better in the unrotated cases, which confirms that jitter exploits the separability of the functions. It is notable, however, that while being able to clearly outperform DITH and AJIT with the unrotated functions, JITT is still able to achieve comparable performance with the rotated versions. The performance differences between different DELL versions are generally smaller than differences between DELL and DEGS. Among the DELL versions, JITT performs the best overall in terms of SP. When the separable cases are left out, the differences in performance between the algorithms are quite small. AJIT has some problems with Schwefel and Rana, but performs well with Rastrigin. Dither, on the other hand, has problems with Rastrigin, but works well with Schwefel.

The use of a randomized scale factor increased the performance in all highly multimodal problems. However, the used set of different NP values was very limited, and it is possible that the performance of DELL using $\sigma = 0$ could be increased by using larger NP values. In most cases, the value $\sigma = 0.1$ offered better robustness than smaller values. Generally DITH seemed to prefer larger σ values than JITT and AJIT. A possible explanation for this is the fact that dither is able to produce a more limited variety of different trials, and needs larger σ value to compensate for the loss. The experiments with $\sigma = 0.5$ suggest that especially with dither values even larger than $\sigma = 0.1$ could be useful in some cases.

The poor performance when using $PX = 0$ was to be expected, because local mutation alone cannot escape local optima efficiently. Disabling local mutation (using $PX = 1$) performs well in some functions, especially in Rana. Still, in most tested cases, a smaller PX value seems to find the optimum more reliably, especially when the dimensionality is increased. The use of local mutation is able to most clearly improve the performance of DELL in the Rosenbrock function. This is well in line with the supposed role of local mutation operation to speed up the local search of the niches. It is logical that the benefits are most clearly visible in Rosenbrock, which is almost unimodal. Because the local mutation seems to be useful in most problems, while not clearly harmful in any of them, using $PX = 1$ is not recommended for initial guess when using DELL. When the extremes are left out, the value of the PX parameter does not seem to be of critical importance to the performance of the DELL. For this reason, a safe initial guess for the value would be around 0.5.

The population sizes used in the tests were close to optimum for Rosenbrock according to the scaling study (Section 5.3.1). However, the harder multimodal problems did not scale similarly, and as dimensionality was increased, often the best performance was achieved with the largest tested population size. For DELL, at the beginning the values of $NP = 3D$ often performed well, but when the dimensionality increased, $NP = 9D$ was required. This suggests that the optimal population size did not increase linearly with the dimension any longer, like in the unimodal problems, but also for DELL the optimum population size will grow faster in multimodal problems. For this reason, it is possible that the use of larger population sizes for the higher dimensional cases would increase SR and SP. As a conclusion, the optimum population size is problem-dependent.

5.4 Niching with Differential Evolution

Several variants of DE using niching methods to extend the usability of the algorithm to multimodal optimization have been presented in the literature. Most of the methods fall in the category of explicit PN, but also the implicit PN concepts of sharing and crowding have been used. This section introduces different DE versions developed for multimodal optimization.

5.4.1 Sharing DE

The fitness sharing concept has been imported to DE by Thomsen [163]. The sharing DE (SHDE) uses the standard sharing function described by equations 3.1 and 3.2, using the Euclidean distance as the distance measure. The method allows the population to double each generation. After *NP* trials have been generated, the sharing function is used for calculating the fitness for each individual, and the worst half of the population is purged. The algorithm ensures elitism by always preserving the individual with best unscaled fitness. The algorithm suffers from the characteristic problems of sharing: the requirement to define σ_{rad} and the high CPT of $O(NP)$.

5.4.2 Crowding DE

Crowding-based DE (CRDE) presented by Thomsen [163] modifies the selection phase of DEGS (eq. 5.3) so that instead of $\vec{x}_i$, the trial $\vec{u}_i$ is compared to the most similar member from a random subset of population $\vec{x}_k$

$$\vec{x}_k = \begin{cases} \vec{u}_i & \text{if } f(\vec{u}_i) \leq f(\vec{x}_k) \\ \vec{x}_k & \text{otherwise} \end{cases}, \tag{5.12}$$

whose size is defined by the *CF* parameter. Also the replacement is done instantly to the main population instead of using a separate trial population. The similarity is measured by the Euclidean distance. To eliminate the additional parameter and URE, Thomsen uses $CF = NP$, which means that CPT is high $O(NP)$. The advantage of CRDE is the lack of additional control parameters. Thomsen performed an experimental study between the sharing and crowding versions of DE. The results demonstrated CRDE to outperform SHDE in all tested cases in locating and maintaining multiple optima.

5.4.3 Speciation-based DE

Speciation-based DE (SDE) proposed by Li [85] is an explicit PN method. The DE population is classified into species according to their similarity measured by Euclidean distance. A procedure for determining species and the dominant individuals in these species is used as in [84]. Each species and its corresponding species seed (the dominant individual) form a separate subpopulation where DEGS is executed locally. To eliminate redundant individuals, a trial having fitness identical to the species seed is replaced by a randomly generated individual. Because species are adaptively formed around different optima, over successive iterations, multiple global optima can be found in parallel. The CPT of $O(c)$ for SDE is similar as in the clearing approach and thus lower compared to the SHDE and CRDE.

As with SHDE, the σ_{rad} parameter is required in order to define the size of a species, which is a major limitation for the method. Parameter m defines the minimum number of members in each species in addition to the species seed. If the species does not have enough members, new individuals are randomly generated within σ_{rad} and added to the species. Between generations, only *NP* fittest individuals are preserved. This includes a risk of losing some potential niches before they have enough time to converge, especially with small population sizes combined with large values of m and small σ_{rad}.

5.4.4 Multipopulation DE

Zaharie [184] proposes a multiresolution multipopulation DE (MMDE), which divides the population to c equally sized subpopulations. The search space is initially divided into c non-overlapping subdomains, for which the subpopulations are initialized. While the DE algorithm is run independently for each subpopulation, the subpopulations are not restricted to the subdomains used in the initialization. The search is divided into epochs, between which the subpopulations are re-initialized using a finer discretization of the domain, so that the number of subdomains increases by c after each epoch. The best solution in each subpopulation is stored in an archive after each epoch. To prevent redundant solutions from entering the archive, the Euclidean distance between each new entree is calculated for each existing solution in the archive, and too similar solutions are discarded. Additionally, hill-valley detection [169] is used to exclude solutions belonging to the same peak. For each subpopulation, a subdomain for re-initialization is randomly selected, but the sharing concept is used to exclude the subdomains for which the archive already contains solutions. MMDE does not require the definition of the σ_{rad} parameter, but introduces a set of new parameters, the number and size of the subpopulations, as well as the number and length of the epochs. The added complexity of the method comes from the re-initialization and archive updating procedures performed between epochs. The complexity increase from the Euclidean distance calculations is affected by the length of an epoch as well as the size of the archive. Thus the CPT for MMDE is approximately $O(n)$, where n defines the number of optima the problem contains, which relates to the size of the archive. However, the hill-valley detection used in updating the archive also requires excess function evaluations, the number of which is related to the archive size and c.

In [183] Zaharie adds crowding to the multipopulation concept producing a multipopulation crowding DE (MCDE). The subpopulations are initialized using subdomains as in MMDE. For each subpopulation, the CRDE is used. As CRDE allows each subpopulation to locate several optima, the epochs and re-initialization of MMDE are no longer needed.

5.4.5 Other approaches

Rigling and Moore [126] have implemented an explicit PN method for DE by tagging population members to belong to different subpopulations and implementing a mating restriction, so that the variation operations are performed only inside each subpopulation. Additionally, a penalty is applied to members of each subpopulation that are too close to members of different subpopulations to drive each subpopulation towards a different optimum. The method requires the definition of the number of subpopulations (c), a penalty term and the minimum spanning distance to be maintained between the subpopulation (effectively the σ_{rad} parameter), which are all problem-dependent parameters and thus form a major difficulty for the use of the method. Rumbler and Moore [131] have tried to overcome these limitations by suggesting a *NewEDE* method for determining the optimal values for these parameters. The idea is to simply run the algorithm repeatedly with different parameter setups to determine suitable values and at the same time keep record of the found solutions. While single values for the problematic parameters are no longer needed, the problem of parameter selection still remains, as the ranges to be tested are still needed. Of course the repeated runs increase the computational complexity.

Hendershot [62] has further extended the NewEDE concept in an algorithm called MultiDE. The algorithm is similarly run with varying values for the c and σ_{rad} parameters and records the best found solutions. The method, however, does not use the penalty term and instead compares each trial to the elements in the recorded optimal solutions, and if it differs less than the value of a precision parameter defined by the user, the trial is discarded. Additionally, for each trial the Euclidean distances to the members of other subpopulations are calculated, and if the trial is within the σ_{rad} from any of these, it is moved away from that subpopulation. The process is repeated until the trial is not within the σ_{rad} from any member of the other subpopulations. To increase the convergence speed, subpopulations which fail to locate new optima within the user-specified maximum number of computations, are eliminated. The main drawback of the MultiDE is the difficulty to tune the control parameters: while the penalty term is removed, the limits for σ_{rad} and c are still required. Additionally, the method adds the precision parameter and the maximum number of computations allowed before elimination of subpopulation to the list of user defined-parameters. The complexity of the method is high, as for each trial $n \cdot (NP - NP/c)$ (all c subpopulations are of equal size) Euclidean distance calculations are required, where n defines the number of repetitions until a suitable trial is found. This gives the method the CPT of $O(NP)$.

5.5 Using DELL for multimodal optimization

This section contains an initial investigation, originally published by the author in [136], of using DELL for multimodal optimization. As the local selection concept by definition isolates each population member, it can be seen as a niching method. Basically local selection functions similarly to the deterministic crowding in the context of GAs, as in both approaches the offspring is pitted against its own parent. Local selection shares the advantages of deterministic crowding: a low CPT of $O(1)$ and the lack of additional control parameters. Additionally, as in DE, only one parent is used, and local selection does not even need to define a similarity measure, which is used in deterministic crowding for determining which of the parents the offspring is compared to. The investigation studied the performance of AJIT using a set of two-dimensional multimodal test problems. The aim was to locate all global optima. Similar runs were performed with mutation-only DEGS to demonstrate the differences between local and global selection. In all cases, 100 independent optimization runs were performed. The results demonstrate the capability of DELS-based approaches in finding multiple global optima, but also point out shortcomings in the original DELL described by Algorithm 3 in the context of multimodal optimization. An improved version of the algorithm is presented in Section 5.5.3. The complete set of related results is available at `http://www.it.lut.fi/ip/evo/`.

5.5.1 The ability to find and maintain optima

The purpose of the first problem set was to test the ability to locate and maintain multiple optima with $\varepsilon = 10^{-3}$ when given more than enough time to converge ($NFE_{max} =$

$NP \cdot 10000$), and to test how the control parameters affect the number of found optima. Three sinusoid based two-dimensional functions were created for the task:

$$f_{s1}(\vec{x}) = \sum_{i=1}^{2} \sin(6.5x_i) \tag{5.13}$$

$$f_{s2}(\vec{x}) = \sum_{i=1}^{2} \sin(20 \cdot \sqrt{x_i}) \tag{5.14}$$

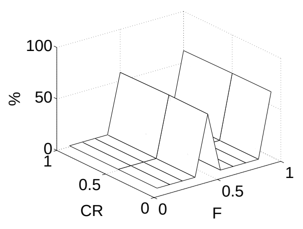
$$f_{s3}(\vec{x}) = \sum_{i=1}^{2} \sin(10\sin(x_i) + 10 \cdot \sqrt{x_i}) \tag{5.15}$$

where $\vec{x} \in [0, 10]^2$. All functions contain 100 global optima and f_{s3} also contains 44 local optima. f_{s1} is regular, while f_{s2} and f_{s3} are partially regular functions. All of the functions are separable. Similar tests were run using randomly rotated versions of the three functions. As the results were almost identical with the unrotated versions, they are not separately reported on here. For the control parameters, the values of $F = \{0.1, 0.2, \ldots, 1\}$, $CR = \{0, 0.5, 1\}$ for DEGS and $\sigma = \{0, 0.1, 0.2, 0.3\}$, $PX = \{0, 0.1, \ldots, 1\}$ and $F = 1$ for AJIT were tested. $NP = \{100, 300\}$ were used for both methods.

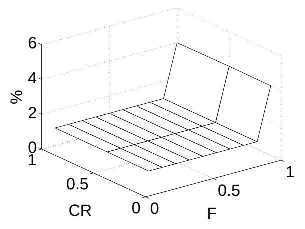
Figure 5.11 displays the results for the first test set. It can be seen that the effect of population size is straightforward: increasing the NP increases the number of found optima. The values of other control parameters do not appear to correlate with the value of NP for either method.

As expected, DEGS has problems in maintaining more than a single optimum. The value of CR has a negligible effect on performance, as the ability to exploit separability does not help in maintaining multiple optima. However, when $F = 1/n$ (where n is an arbitrary integer), DEGS performs very well in f_{s1}. Also in $f_{s2}(\vec{x})$ and $f_{s3}(\vec{x})$, DEGS is able to locate and maintain more than a single optimum with $F = 1$. The results can be explained by the ability of DEGS to exploit the regularity of the functions. As the algorithm starts to converge, the population begins shaping up around the optima. Now any unscaled differential between different optima is ideal for jumping from an optimum directly to another in regular functions, where the optima are equally spaced. Once the population has located a couple of optima, DEGS will quickly locate the rest by just directly jumping to them. This also negates the effect of global selection, which tends to lose already located optima over time, because even if some of the optima are lost, the information on how to jump there from another still exists in the population and the lost optimum will be found shortly. The effect is strongest with $F = 1$, but any value $F = 1/n$ allows such direct jumps, as long as the function has at least $n + 1$ equally spaced optima in a row. For example with $F = 0.5$, a differential suitable for jumping directly from one optima to another is only between optima which have odd number of optima between them. The jumping does not work as well in f_{s2} and f_{s3}, which are only partially regular. However, using $F = 1$ allows DEGS to exploit even the partial regularity to locate more than one optimum.

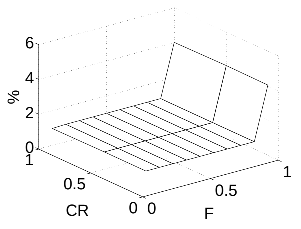
AJIT was able to locate a good portion of the optima with all parameter setups. In f_{s1} DELL achieved best performance with $\sigma = 0$ and $PX > 0$, because with that setup



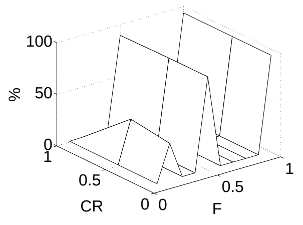
(a) DEGS,f_{s1},NP=100



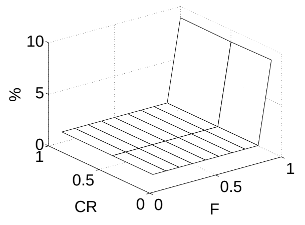
(b) DEGS,f_{s2},NP=100



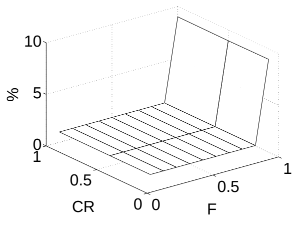
(c) DEGS,f_{s3},NP=100



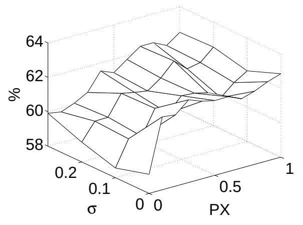
(d) DEGS,f_{s1},NP=300



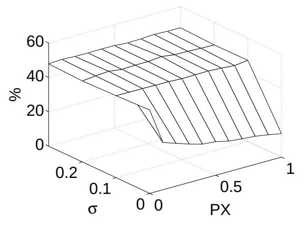
(e) DEGS,f_{s2},NP=300



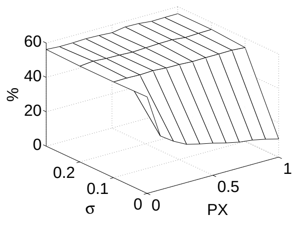
(f) DEGS,f_{s3},NP=300



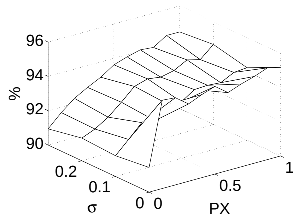
(g) AJIT,f_{s1},NP=100



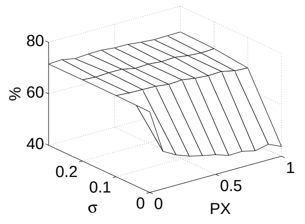
(h) AJIT,f_{s2},NP=100



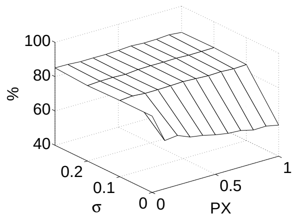
(i) AJIT,f_{s3},NP=100



(j) AJIT,f_{s1},NP=300



(k) AJIT,f_{s2},NP=300



(l) AJIT,f_{s3},NP=300

Figure 5.11: Average percentage of found global optima. Note the different scales.

it could exploit the regularity of the function similarly as DEGS through the global mutation. Still, the values of parameters PX and σ had only a small effect on the number of located optima. In f_{s2} and f_{s3}, AJIT achieved a steady number of optima as long as $\sigma > 0$. With $\sigma = 0$, the performance dropped rapidly when global mutation was allowed.

5.5.2 Convergence speed

The second test set consisted of four well known two-dimensional multimodal test functions implemented inside the common family of the generator framework (Section 4.4.5): the Himmelblau, Six-hump camel back, Branin and Shubert functions. The problems were selected to provide results comparable with the ones presented in [183] and [85] for other DE-based multimodal optimization approaches. For this reason, the functions were not stretched or rotated. The average NFE required to find all global optima with $\varepsilon = 10^{-4}$ ($\varepsilon = 10^{-3}$ for f_{shu}) were recorded using $NFE_{max} = NP \cdot 2000$. For the calculation SR and SP, the desired set consisted of all the global optima of each function. The control parameter values were: $NP = \{3 \cdot NGO, 5 \cdot NGO, 10 \cdot NGO, 15 \cdot NGO\}$, $F = \{0.5, 1\}$, $CR = 1$ for DEGS and $\sigma = \{0, 0.1, 0.2\}$, $PX = \{0, 0.5, 1\}$ and $F = 1$ for AJIT.

Table 5.2 shows the most interesting results for the second test set, including the best achieved SP and the maximum SR with minimum NFE for both methods. Additionally, some results reported in the literature are included for comparison. AJIT is able to achieve $SR = 1$ in all tested problems, while DEGS has problems in locating all the global optima regularly with the first three functions, as expected. $F = 1$ works best but also with $F = 0.5$ DEGS is able to, at least occasionally, find all optima for all problems except f_{him}. The comparable results for the first three functions taken from the literature are of a similar magnitude as the results for AJIT. While AJIT is the slowest niching approach in f_{him} and f_{shcb}, it outperforms SDE in f_{bra}.

In Shubert, DEGS has no problems at all and is able to solve the problem faster than any of the methods using niching when $F = 1$, as it exploits the regularity. As expected, DELL gives best results in Shubert with $\sigma = 0$ and $PX = 1$, which allow the algorithm exploit the regularity efficiently. To find all the optima with DELS using $\sigma > 0$, AJIT requires almost ten times more function evaluations. DEGS with $F = 0.5$ also performs well, requiring clearly fewer function evaluations compared to AJIT. For comparison, MCDE using two subpopulations and $F = 0.5$ is about 50% faster than DEGS with $F = 0.5$, as can be seen in table 5.2. CRDE similarly using $F = 0.5$ is, however, clearly slower. It is logical that the crowding procedure slows down the convergence compared to DEGS in regular functions, but the good performance of the subpopulation approach is somewhat surprising, as the division to subpopulations decreases the ability of an algorithm to exploit the regularity simply because fewer regularities exist within the subpopulation. The explanation for the good performance of MCDE probably lies in the fact that also the local optima are regularly spaced. Using a small number of subpopulations thus does not fully destroy the ability to exploit the regularities, but allows smaller population sizes, which increases the convergence speed. The result for MCDE is still over three times slower than the best result with unmodified DEGS using $F = 1$, and over two times slower than with DELL using $\sigma = 0$ and $PX = 1$. The result for MMDE with Shubert uses 20 subpopulations and is unable to exploit the regularity. The performance of MMDE is in the same magnitude class as AJIT with $\sigma > 0$ and indicates the NFE required to solve the problem without exploiting the regularity.

Table 5.2: Results for the second test set on finding all global optima.

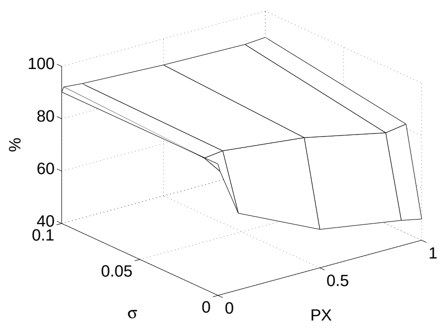
$f(\vec{x})$	Method	NP	PX or F	σ or CR	$NFE \pm std$	SP	SR	Source/comments
f_{him}	AJIT	40	1	0.1	8744 ± 7396	8744	1	
	AJIT	20	0.5	0.2	4143 ± 894	6183	0.67	
	DEGS	60	1	1	5449 ± 552	49537	0.11	
	SDE	50	0.5	0.9	5236 ± 5166	5236	1	[85]
	CRDE	20	0.5	0.9	8033	8033	1	[183],$\varepsilon = 10^{-6}$
	MCDE	20	0.5	0.9	7486	7486	1	[183],$\varepsilon = 10^{-6}$
	MMDE	50			6069	6069	1	[183],$\varepsilon = 10^{-6}$
f_{shcb}	AJIT	20	0.5	0.1	1100 ± 205	1100	1	
	AJIT	10	1	0.2	645 ± 130	679	0.95	
	DEGS	20	1	1	738 ± 101	777	0.95	
	DEGS	30	0.5	1	788 ± 154	1487	0.53	
	SDE	50	0.5	0.9	723 ± 124	723	1	[85]
f_{bra}	DELL	45	1	0	3733 ± 398	3733	1	
	AJIT	30	0.5	0.1	3733 ± 537	3733	1	
	AJIT	15	1	0.1	2202 ± 874	2591	0.85	
	DEGS	30	1	1	1904 ± 205	3592	0.53	
	DEGS	45	0.5	1	2143 ± 212	13395	0.16	
	SDE	50	0.5	0.9	4360 ± 2799	4360	1	[85]
f_{shu}	DELL	90	1	0	18823 ± 1215	18823	1	
	AJIT	180	0.5	0.1	156010 ± 10840	156010	1	
	DELL	54	1	0	12535 ± 1844	12661	0.99	
	AJIT	90	0.5	0.1	91542 ± 13181	108980	0.84	
	DEGS	54	1	1	8141 ± 943	8141	1	
	DEGS	180	0.5	1	46753 ± 6566	60719	0.77	
	CRDE	75	0.5	0.9	107200	107200	1	[183]
	MCDE	50	0.5	0.9	29283	29283	1	[183]
	MMDE	400			109703	109703	1	[183]

5.5.3 Improved DELL version for multimodal optimization

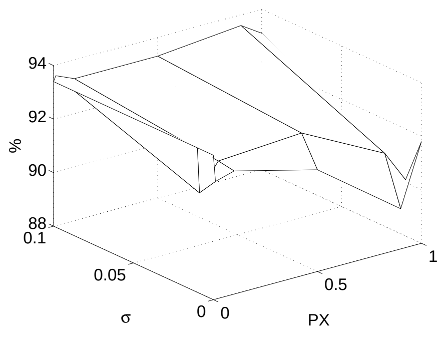
The results of the tests (Figure 5.11) clearly demonstrate the inability of DELL to maintain multiple global optima effectively in other than regular functions without using a randomized scale factor, i.e. when using $\sigma = 0$. The reason for this effect is that Algorithm 3 allows the target vector to be selected as either of the random vectors ($r_1 = i$ or $r_2 = i$) used to calculate the differential. With $\sigma = 0$, the trial may become a copy of the other individual participating in the differential calculation. Because the trial is always selected over the target if they have the same objective function value, this allows jumps directly from one optimum to another. If the target is the only member residing in that particular optimum, the optimum will be lost. By preventing the target from participating in the calculation of the differential ($r_1 \neq i$ and $r_2 \neq i$), such jumps become very unlikely, unless the optima are regularly spaced and the function contains several differentials of equal length between the optima. In such cases, however, losing an optimum is usually temporary, because the differential to jump back is still stored in the population, and eventually the population tends to settle on all the optima. Algorithm 4 defines the improved DELL version, for which the abbreviations DELL, AJIT, JITT and DITH are used to refer for the rest of this thesis. An example of the effect of the modification is demonstrated in Figure 5.12. It can be seen that the percentage of found optima drops dramatically with the original version when $\sigma = 0$ is used, compared to the modified version.

Algorithm 4 DELL, improved version for multimodal optimization

```
1:  Initialize population, $g = 1$
2:  while termination criterion not met do
3:    for $i = 1; i \leq NP; i = i + 1$ do
4:      Randomly pick $r_1, r_2 \in \{1, 2, \ldots, NP\}, r_1 \neq r_2 \neq i)$
5:      if $rand[0, 1] < PX$ then
6:        Perform global mutation using (5.8),(5.9) or (5.10)
7:      else
8:        Perform local mutation using (5.7)
9:      end if
10:   end for
11:   for $i = 1; i \leq NP; i = i + 1$ do
12:     Perform selection using (5.3)
13:   end for
14:   $g = g + 1$
15: end while
```



(a) Original



(b) Improved

Figure 5.12: Average percentage of found optima from 100 runs on $f_{q2,0}$ (see Section 6.1.1), using $NP = 100$ and AJIT with varying PX and σ. Note the different scales.

5.6 Hybridization

Several approaches for hybridizing DE with local search methods to increase the convergence speed have been presented in the literature. Tirronen et al. [165, 164] propose a memetic differential evolution approach. The method runs DE and monitors the population diversity. After the population diversity decreases below a certain threshold, the algorithm periodically activates a stochastic local search for a predetermined number of function evaluations using a randomly selected population member. Similarly, the Hooke-Jeeves algorithm is activated to be run for the best performing population member when the diversity falls below a lower threshold. The algorithm has been enhanced in [166] by adding simulated annealing as a fourth method to the hybrid. Now SA has the highest activation threshold and it is used for a random population member. When the diversity falls further, the Hooke-Jeeves gets activated using the best performing individual, then stochastic local search using random, and finally using the best performing individual. *Super-fit memetic Differential Evolution* [20] follows a similar philosophy. Now PSO is first run for part of the population prior to DE to quickly generate a highly fit solution, which is then included to the DE population. The actual hybrid first activates the Nelder-Mead simplex algorithm using random population members, and when the diversity decreases further, the Rosenbrock algorithm using the best performing individual.

Gao and Wang [47] also hybridize the DE with the Nelder-Mead simplex. The algorithm first runs DE for a preset number of iterations and then runs the Nelder-Mead simplex from the best located point. After that the worst half of the population is re-initialized and the search process is repeated. Chiou and Wang [25] suggest acceleration and migration operators to DE. The acceleration operation speeds up the local search if no improvement is observed between the generations. It replaces the worst population member using a candidate generated from the best member by jumping to gradient direction and scaling the step size accordingly. The algorithm also monitors population diversity and uses the migration operator to generate a new population, when the diversity drops too low. The new points are generated by adding normal distributed random vectors to the best member of the previous population.

Noman and Iba [110] propose a *fittest individual refinement* strategy, which hybridizes DE with XLS. A local search step is used each generation by generating a predetermined number of points around the best population member in the hope of improving it further. The method is developed further in [111] by using adaptive hill-climbing, which takes feedback from the search for determining the step length of LS.

5.6.1 Hybrids for multimodal optimization

To prevent premature convergence or to offer a set of good starting points for local search in post-hybridization methods, memetic algorithms often implement different niching methods to keep the population diverse. For example Wei and Mei [173] hybridize GA with the Nelder-Mead simplex method, using clearing for niching. Zhang and Lu [186] similarly propose a GA hybrid using a modified simplex technique. Niching is done by using a dynamic subpopulation strategy, which divides the population into subgroups. Peng et al. [115] hybridize GA with a gradient descent and use the dynamic niche sharing algorithm for applying a mating restriction. While such hybrids using niching would be potential approaches for multimodal optimization, this aspect is rarely considered. An exception are Ono et al. [112], who hybridize a sharing GA as well as an immune algorithm with a Quasi-Newton local search method to find multiple optima. Their results suggest that hybridization is able to improve the performance, compared to the parent algorithms.

5.6.2 Proposed hybrid algorithms

The two hybrid algorithms proposed in this section have been previously published by the author in [138]. The idea to differentiate global and local phases is already used in DELL. The local mutation step length is based on the results of DELS performance on unimodal problems. In such cases, the population converges towards a single solution, which speeds up the convergence when the average differentials shorten.

In multimodal optimization the situation changes, because the population converges slower and towards multiple attractors, which keeps the average differentials longer. Thus the used formula for calculating the step length in equation 5.7 may produce larger than necessary values for an efficient local search. The convergence speed may be modified by changing the step length, but this would create another control parameter, and finetuning the mutation step length is not likely to make the operator match the speed of pure local search approaches. For this reason, a new DELS based algorithm using gradient descent for local search (*DELG*) instead of local mutation is proposed. The reason for selecting gradient descent over the more efficient conjugate gradient method [121, p. 420], is that gradient descent does not require the storing of information between line searches. The proposed hybrid method is described in Algorithm 5. Each time the local phase is selected, the algorithm performs a single line search in the direction of the approximated gradient, not a complete local search. Thus, the global and local search advance in parallel, their relative emphasis decided by PX. With $PX = 1$, DELG and DELL are identical, and with $PX = 0$, DELG simply performs a local search for each population member. The abbreviation *DITG* refers to dither, *JITG* to jitter and *AJIG* to the additive jitter version of DELG. DELG shares the advantages of DELL: low CPT of $O(1)$ and lack of added problem-dependent control parameters.

Algorithm 5 DELG

```
1: Initialize population, $g = 1$
2: while termination criterion not met do
3:   for $i = 1; i \leq NP; i = i + 1$ do
4:     Randomly pick $r_1, r_2 \in \{1, 2, \ldots, NP\}, r_1 \neq r_2 \neq i)$
5:     if $rand[0,1] < PX$ then
6:       Perform global mutation using (5.8), (5.9) or (5.10)
7:     else
8:       Set $\vec{x}_a = \vec{x}_{i,g}$
9:       Calculate normalized gradient $\vec{g}_n = \vec{g}/|\vec{g}|$
10:      Perform bracketing using method presented in [121, p. 400], using initial
         points $\vec{x}_a$ and $\vec{x}_b = -s \cdot \vec{g}_n + \vec{x}_a$
11:      Perform line search using method presented in [121, p. 404]
12:    end if
13:  end for
14:  for $i = 1; i \leq NP; i = i + 1$ do
15:    Perform selection using (5.3)
16:  end for
17:  $g = g + 1$
18: end while
```

The local selection-based approaches were initially designed for locating multiple global optima, but their ability for preserving also local optima is yet to be verified. CRDE, on the other hand, has been shown to be able to preserve local optima reasonably well [183, 163]. Using the structure for DELG, the implementation of DE algorithm using crowding and gradient descent (*DECG*) is straightforward. The algorithm similarly uses global mutation for global and gradient descent for the local search phase. Instead of using the local selection where the trial is compared to its own parent, however, the trial is compared to the most similar member of the populations, similarly as in CRDE. The modified algorithm introduces no additional parameters compared to DELG (assuming $CF = NP$ is used). The downside of the method compared to DELG is that the use of crowding requires *CF* euclidean distance calculations for each generated trial, and thus CPT is $O(NP)$ as in CRDE. Algorithm 6 describes DECG.

Algorithm 6 DECG

```
 1: Initialize population
 2: while termination criterion not met do
 3:   for $i = 1; i \leq NP; i = i + 1$ do
 4:     Randomly pick $r_1, r_2 \in \{1, 2, \ldots, NP\}, r_1 \neq r_2 \neq i)$
 5:     if $rand[0, 1] < PX$ then
 6:       Perform global mutation using (5.8), (5.9) or (5.10) ignoring the subscript $g$
 7:     else
 8:       Set $\vec{x}_a = \vec{x}_i$
 9:       Calculate normalized gradient $\vec{g}_n = \vec{g}/|\vec{g}|$
10:       Perform bracketing using method presented in [121, p. 400], using initial
          points $\vec{x}_a$ and $\vec{x}_b = -s \cdot \vec{g}_n + \vec{x}_a$
11:       Perform line search using method presented in [121, p. 404]
12:     end if
13:     for $j = 1; j \leq NP; j = j + 1$ do
14:       Calculate Euclidean distance between $\vec{u}_i$ and $\vec{x}_j$. The population member
          with the shortest distance is $\vec{x}_k$
15:     end for
16:     Perform selection using (5.12)
17:   end for
18: end while
```

DELG and DECG offer an interesting pair of DE-based hybrid algorithms for multimodal optimization. Both offer effective separation of global and local search phases. DE using global mutation allows the algorithm to identify and exploit the global features of the optimized function, while the local search method ensures efficient local convergence behavior. As the local and global search are performed in parallel, the decision when to switch between the algorithms can be avoided. Instead, the user decides a value for the *PX* parameter, which controls the emphasis towards the global or local search. The difference between the approaches is that DELG allows multimodal optimization with minimum added computational effort. DECG, on the other hand, will potentially increase the ability to maintain more optima, but with added computational cost.

5.7 Summary

This chapter has concentrated on the Differential Evolution algorithm. First the connection between the crossover and the rotational invariance of the algorithm was described, as well as the stagnation problem. As a response for the stagnation, randomization of the mutation scale factor was used. The author's own study of the effects of jitter revealed that using large values for the σ decrease the performance of the algorithm, but with appropriately small σ values, the randomization may be beneficial in some functions. The jitter concept was developed further by introducing an additive jitter operator, which otherwise behaves like jitter, but is a rotationally invariant process.

Global and local selection were compared in a scaling study using simple unimodal functions revealing a strong tendency of the global selection for drawing the population to uniformity. Local selection does not have such tendency, which allows it to be used as a niching method. A new DE algorithm for multimodal optimization based on the local selection concept was constructed by dividing the mutation operation to local and global parts, so that the local part is responsible for speeding up the local search while the global part identifies and exploits global function features. The idea of randomizing the scale factors was also incorporated in the global mutation. Two studies comparing DELS and DEGS algorithms on multimodal functions were conducted. The first concentrated on the ability of the algorithms to locate a single global optimum and the second to their ability to do multimodal optimization. DELS demonstrated promising results in both respects. An overview of different niching methods in the literature used with DE, as well as a limited experimental comparison to DELS was also included.

Finally, the topic of using hybridization to improve the performance of multimodal optimization methods was considered. The idea of separating the local and global search phases was taken further by presenting two hybrid algorithms for multimodal optimization combining DE and a gradient descent local searcher. DELG uses the local selection for niching while DECG relies on crowding. The next chapter presents experimental studies comparing different DE-based multimodal optimization algorithms, including the methods proposed in this chapter.

Chapter VI

Experiments

This chapter presents experimental results of using eight different multimodal optimization approaches on a set of problems generated by the test function generator described in Section 4.4. The results have been previously partially published in [138]. The roles of local and global search in multimodal optimization are examined through studying different approaches and their performance in different problems. In addition to identifying the best performing methods for each problem type, the reasons behind the performance differences are highlighted to increase the understanding of the behavior of different optimization approaches. The focus is especially on studying the effect of the degree of regularity of the problem, which can be seen as an indication of the ability of an optimization method to exploit global information. In addition, the effects of increasing the number of optima and dimensionality of the function are studied. The ability of an algorithm to maintain stable niche populations is studied by including a set of problems, where also local optima are of interest. Many niching methods bring along additional control parameters, for which good values are often problem-specific and may prove hard to tune correctly. For this reason, an extensive parameter study is performed for each method.

Eight different approaches are compared on the basis of their ability to locate global optima. DEGS is used as a benchmark and base model for all evolutionary methods. The use of a single base model allows the differences between niching methods to become visible. CRDE, SDE and DELL are used as representatives of different DE-based niching methods. Two multistart gradient descent methods, RSGD and GBGD are included to offer a baseline for comparison. In addition, two proposed hybrid approaches, DELG and DECG are studied. Configuration files and example figures for each function, along with a comprehensive set of plotted result curves are available at: `http://www.it.lut.fi/ip/evo/`. In all cases, constraint handling is done using the mirroring approach provided by the generator module described in Section 4.4. To make the functions nonseparable, and thus eliminate the need to consider values smaller than $CR = 1$ for DE, all functions from the cosine and common families are randomly rotated. Bezier curves of tenth degree with random control points are used in all stretched cases. For all tests, 100 independent runs are performed, so that each time a different random seed from $0, 1, \ldots, 99$ is used to initialize the generator. Thus each run is done with a different function. This is done to minimize the ability of an algorithm to exploit a specific feature of a single function instance and to concentrate on the behavior of the algorithms with function types. The same function set is always used for each algorithm. The performance is determined primarily by the average PR and secondarily by the speed to locate the i:th global optimum measured as NFE using $\varepsilon = 10^{-4}$. For the GD methods, new line searches are performed until the improvement line search is able to provide is smaller than 10^{-6}.

6.1 Locating multiple global optima

The first test setup compares the ability of the eight algorithms to locate all global optima within $NFE_{max} = D \cdot 250000$ function evaluations. For all DE-based methods, crossover is always disabled by using $CR = 1$ to keep the algorithms rotationally invariant. For the population size, the best performing value among $NP = [50, 100, 200, 300, 500, 1000, 2000]$ is searched for each problem and each method individually. A fixed $PX = 0.5$ for DELL and $PX = 0.9$ for DELG and DECG are used in all reported results comparing algorithms. DELL is not sensitive to the value of the PX parameter, as long as the extremes (0 and 1) are avoided, which would disable either the local or global mutation. For the hybrids, larger PX has been selected to prevent the algorithms from becoming overly greedy. Similarly, all DELS-based algorithms use a fixed $F = 1$ and $\sigma = 0$ for the global mutation to keep the mutation step length global and to disable randomization. For CRDE and SDE, the best performing value for mutation step length among $F = [0.1, 0.5, 0.9, 1]$ is searched for each function due to the difficulties of finding a well-performing value for all problems. For DEGS, similar values have been tested, but the fixed $F = 1$ demonstrate the best performance in all functions, and thus it is used in all comparisons. Additionally, a best value for σ_{rad} is searched for SDE among $\sigma_{rad} = [0.05, 0.5, 2]$ for the common family functions, among $\sigma_{rad} = [0.005, 0.05, 0.2]$ for the other two-dimensional functions, among $\sigma_{rad} = [0.05, 0.08, 0.2]$ for the five-dimensional cases and among $\sigma_{rad} = [0.05, 0.11, 0.2]$ for the ten-dimensional functions. Fixed $m = 10$ is used to decrease the number of tunable parameters of SDE as suggested in [85]. Some parameter combinations have been left untested for some methods and functions, due to existing results and assumptions of their poor chance of improving the results.

The idea of searching good parameter combinations for each algorithm is to make the comparison fair in the sense that it reveals the true potential of each algorithm and shows how sensitive they are to the control parameters. The setup is actually somewhat tilted to the favor of CRDE and SDE, as most of their parameters are optimized, while for DELL, DELG and DECG only the best population size is sought for each problem, and the other parameters are fixed as they are less problem-dependent. However, some tests with DELS-based methods are performed using different values of the PX, as well as values of $\sigma > 0$ with both dither and ajitter versions of global mutation to study the effects of changing parameters and the sensitiveness of the algorithms to these parameters. These results are not included in the comparisons between different algorithms, to keep the results directly comparable to the ones acquired with DECG.

6.1.1 Function setup

The setup consists of four regular, five partially regular and seven irregular functions. The regular functions are the Shubert function and three functions from the cosine family. To obtain partially regular functions, all the regular cases are stretched. Additionally, the Vincent function is included. In the cosine functions, the basic shape of the global optima is a 5 by 5 square ($\vec{G} = [5, 5]$ and $\alpha = 0.8$ is used). Among the resulting 25 global optima, 16 are located at the constraints and 9 are inside the search space. The functions differ by the number of local optima (NLO), for which the values $\vec{L} = [1, 1]$, $\vec{L} = [10, 10]$ and $\vec{L} = [30, 30]$ are used, producing 0, 1656 and 14616 local optima, respectively.

The irregular functions are generated using the quadratic family. For all cases, rotated ellipsoidal shapes are used for the optima, so that the values used to generate C are randomly generated in the range $[0.003, 0.03]$, the optimum value of local optima in the range $[-0.95, -0.15]$ (global optima have value -1), and the minimum possible Euclidean distance between two global optima points is set to 0.01. Versions with 0 and 100 local optima are generated for two, five and ten dimensions, each having ten global optima. In addition, a two-dimensional function with 1000 local optima is included. Table 6.1 summarizes the test function setup.

Table 6.1: First function test set.

Function	NGO	NLO	D	Family	Regularity
f_{cL1}	25	0	2	cosine	regular
f_{cL10}	25	1656	2	cosine	regular
f_{cL30}	25	14616	2	cosine	regular
f_{Rshu}	18	742	2	common	regular
f_{ScL1}	25	0	2	cosine	partial
f_{ScL10}	25	1656	2	cosine	partial
f_{ScL30}	25	14616	2	cosine	partial
f_{SRshu}	18	742	2	common	partial
f_{Rvin}	36	0	2	common	partial
$f_{q2,0}$	10	0	2	quadratic	irregular
$f_{q2,100}$	10	100	2	quadratic	irregular
$f_{q2,1000}$	10	1000	2	quadratic	irregular
$f_{q5,0}$	10	0	5	quadratic	irregular
$f_{q5,100}$	10	100	5	quadratic	irregular
$f_{q10,0}$	10	0	10	quadratic	irregular
$f_{q10,100}$	10	100	10	quadratic	irregular

6.1.2 Results and analysis

Table 6.2 presents the best achieved PR results, along with the standard deviations for each tested algorithm in the functions of the first test set. The best result for each function is marked in the **bold font**, as well as any other results which do not have a statistically significant difference to the best one with a significance level 0.05. Similarly, the worst results are marked in the *italic font*. The significance is determined using the two-tailed Wilcoxon signed-rank test [178]. The performance column lists the tested methods from the best to the worst performer in each problem, so that C is CRDE, G means DELG, E stands for DECG, R is RSGD, D means DEGS, S stands for SDE, L is DELL and I is GRGD. In cases with no statistically significant difference in the peak ratios, the speed measured as the number of function evaluations is used to determine the performance order. The brackets are used to group methods when their relative performance order cannot be confidently defined. Table 6.3 displays the parameter setups for each method which produced the results presented in Table 6.2.

Table 6.2: Results for the first function test set.

Function	Performance	DELG	DECG	CRDE	DELL	SDE	DEGS	RSGD	GRGD
f_{cL1}	D(IGR)LSEC	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	std	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
f_{cL10}	(DGL)(ECS)RI	**1.000**	**0.998**	**0.998**	**1.000**	0.997	**1.000**	0.960	*0.914*
	std	0.000	0.008	0.010	0.000	0.010	0.000	0.041	0.049
f_{cL30}	(DGL)(EC)SRI	**1.000**	**0.998**	**0.998**	**1.000**	0.990	**1.000**	*0.378*	*0.360*
	std	0.000	0.009	0.009	0.000	0.021	0.000	0.100	0.000
f_{Rshu}	D(LG)E(SC)IR	1.000	1.000	1.000	1.000	1.000	1.000	*0.994*	1.000
	std	0.000	0.000	0.000	0.000	0.000	0.000	0.017	0.000
f_{ScL1}	(RI)GS(LCE)D	**1.000**	0.996	**0.999**	**0.998**	**0.998**	*0.854*	**1.000**	**1.000**
	std	0.000	0.015	0.006	0.011	0.011	0.154	0.000	0.000
f_{ScL10}	GS(LE)RCID	**0.999**	0.956	0.809	0.964	0.985	*0.714*	0.858	*0.725*
	std	0.007	0.042	0.078	0.069	0.028	0.248	0.089	0.143
f_{ScL30}	GLECSDRI	**0.953**	0.815	0.684	0.871	0.654	0.616	0.291	*0.188*
	std	0.063	0.112	0.120	0.156	0.137	0.254	0.098	0.099
f_{SRshu}	G(LS)CEI(RD)	**1.000**	0.910	0.927	**1.000**	**0.999**	*0.874*	*0.882*	0.896
	std	0.000	0.100	0.153	0.000	0.008	0.151	0.115	0.122
f_{Rvin}	RGICLSED	0.959	0.696	0.901	0.804	0.770	*0.359*	**0.970**	0.956
	std	0.032	0.047	0.035	0.048	0.049	0.044	0.023	0.025
$f_{q2,0}$	(IR)G(SE)LCD	**1.000**	**0.997**	0.988	**0.997**	**1.000**	*0.574*	**1.000**	**1.000**
	std	0.000	0.017	0.036	0.017	0.000	0.147	0.000	0.000
$f_{q2,100}$	(GRIE)(LS)CD	**1.000**	0.992	0.944	0.987	0.985	*0.613*	**0.999**	**0.998**
	std	0.000	0.027	0.066	0.039	0.041	0.128	0.010	0.014
$f_{q2,1000}$	(GE)(LI)RSCD	**0.990**	**0.985**	0.716	**0.983**	0.912	*0.635*	0.948	0.973
	std	0.030	0.036	0.268	0.038	0.091	0.144	0.069	0.053
$f_{q5,0}$	RIE(GSC)LD	0.957	**0.989**	0.954	0.777	0.958	*0.216*	**0.996**	**0.992**
	std	0.062	0.035	0.069	0.136	0.065	0.091	0.020	0.031
$f_{q5,100}$	RIEGLSCD	0.701	0.769	0.211	0.502	0.313	*0.120*	**0.905**	0.853
	std	0.145	0.138	0.129	0.161	0.135	0.085	0.104	0.116
$f_{q10,0}$	REIGCSLD	0.893	0.957	0.728	0.411	0.619	*0.117*	**0.981**	0.932
	std	0.101	0.061	0.222	0.175	0.106	0.038	0.042	0.075
$f_{q10,100}$	R(IE)GLCSD	0.550	0.643	0.205	0.252	0.117	*0.033*	**0.800**	0.667
	std	0.142	0.152	0.151	0.148	0.089	0.051	0.135	0.151

Figure 6.1 presents a classification of the algorithms in regard to their ability to identify and exploit the regularity of the problem versus their convergence speed. The ability to exploit regularity can be seen as an indicator of the ability of an algorithm to exploit global information. The convergence speed illustrates the speed in finding the first optimum in problems with no local optima and can be seen as an indicator of the effectiveness of the local search phase. Because the parameter setups affect the performance, the classification is qualitative, and should be seen as a guideline for the properties of

Table 6.3: Best parameter setups for the first function test set.

Function	DELG	DECG	CRDE		DELL	SDE			DEGS
	NP^*	NP^*	NP^*	F^*	NP^*	NP^*	F^*	σ^*_{rad}	NP^*
f_{cL1}	100	300	100	0.5	100	500	1	0.05	100
f_{cL10}	100	300	100	0.5	100	2000	1	0.2	100
f_{cL30}	100	300	100	0.5	100	2000	1	0.2	100
f_{Rshu}	100	100	100	1	50	500	1	0.5	100
f_{ScL1}	1000	1000	500	0.1	1000	500	1	0.05	1000
f_{ScL10}	1000	500	100	1	500	500	1	0.05	1000
f_{ScL30}	300	100	100	1	200	2000	1	0.05	1000
f_{SRshu}	500	50	100	1	300	1000	1	0.5	1000
f_{Rvin}	2000	2000	500	0.1	1000	2000	0.5	0.05	2000
$f_{q2,0}$	1000	500	100	0.1	1000	200	1	0.005	1000
$f_{q2,100}$	1000	300	100	1	500	200	0.5	0.005	1000
$f_{q2,1000}$	1000	500	100	1	500	100	0.5	0.005	1000
$f_{q5,0}$	300	300	200	1	500	100	0.1	0.2	1000
$f_{q5,100}$	500	500	100	0.1	300	1000	0.5	0.05	1000
$f_{q10,0}$	300	300	100	0.9	200	1000	0.5	0.05	500
$f_{q10,100}$	500	300	100	0.5	300	2000	0.5	0.11	500

the algorithms in typical cases.

The GD approaches are clearly the greediest of the algorithms, because they concentrate solely on local search. RSGD neglects all global information, including regularity. GRGD, on the other hand, aims to divide the points evenly in the search space and can potentially exploit regularity efficiently. The ability is limited to cases where the grid fits well to the optima, however, and can lead to a poor performance if the grid is poorly positioned in regard to the optima.

The hybrids are second in raw convergence speed, losing to pure GD due to the mutation operation and the use of population. On the other hand, these features allow them to direct the search with the global information. Especially using $\sigma = 0$ and $F = 1$ allows direct jumps between regularly spaced optima, and as a result DELG becomes very efficient in regular cases, rivaled only by DELL with $\sigma = 0$ and DEGS with $F = 1$. The crowding hinders the ability of DECG to exploit the regularity, because it hinders the process of convergence to the optima and decreases the number of available differentials suitable to be exploited.

DEGS and SDE are in the middle for the convergence speed comparison. DEGS is naturally greedy for an evolutionary method, because the global selection allows points to jump towards the best found solution, and there is no mechanism for keeping the diversity. SDE performs DEGS locally, covering only a part of the search space at a time.

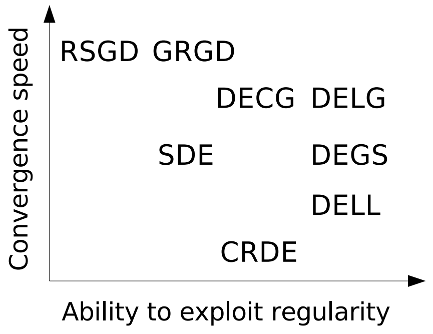


Figure 6.1: Comparison of the algorithms.

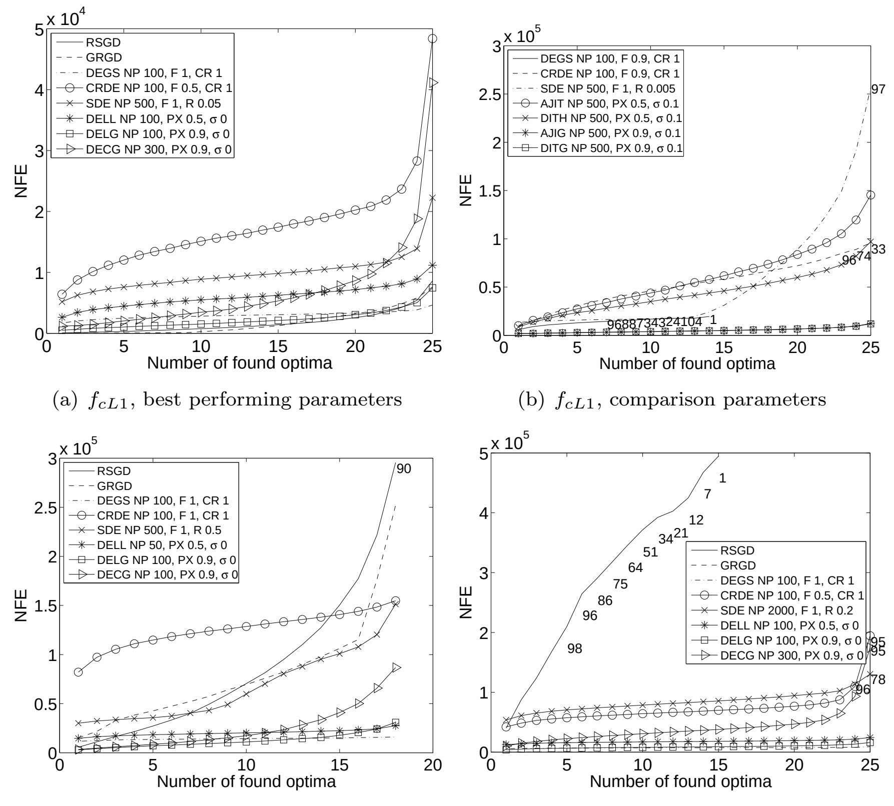
The algorithm is very sensitive to the σ_{rad} parameter, and the value has a significant impact on both, the local and global behavior of the algorithm. Mainly, σ_{rad} limits the maximum differential inside a species, thus affecting the mutation step size. Basically, larger σ_{rad} values allow longer differentials and strengthen the global phase, while smaller values decrease the differentials and take the algorithm towards parallel stochastic local search. Using a large enough σ_{rad} value to include the whole search phase would make the algorithm DEGS. In practice, σ_{rad} must be defined as small enough not to include several global optima, which hurts the ability of the algorithm to exploit regularity.

Compared to the other methods, DELL and especially CRDE are slower. In both, the local search phase is based on mutation. Because they keep the population diverse, the average differentials will stay longer compared to DEGS. This interferes with the idea of self-adaptation of DE, where the local search is attained automatically by the shortening differentials as the population begins to converge. DELL is faster of the two, because a smaller scaling constant is used for the local mutation. This also allows DELL to exploit the regularity through global mutation efficiently, which uses unscaled differentials, independently of the local part. In CRDE, parameter F always defines the scaling. The value $F = 1$ will allow the algorithm to exploit regularity, but hinders the convergence speed, while a smaller value increases the convergence speed, but decreases the ability to exploit global information. Next, we look at the results in functions with varying degree of regularity in closer detail.

Regular functions

Figure 6.2 displays convergence graphs for the different methods on the regular functions. Figures 6.2(a), 6.2(c) and 6.2(d) display the best achieved results corresponding to the PR values in Table 6.2. Figure 6.2(b) highlights some of the most interesting effects of changing parameter setups for different methods.

The best performer in all regular cases is DEGS using $F = 1$, as can be seen in Figure 6.2 and Table 6.2, followed closely by DELG using $\sigma = 0$. Also DELL performs well in cases which contain local optima, but can not match the speed of the fastest methods in the easiest f_{cL1} function. The results are especially interesting in the light that DEGS does not use any niching methods and is thus designed for locating only one optimum. The



(a) f_{cL1}, best performing parameters

(b) f_{cL1}, comparison parameters

(c) f_{Rshu}, best performing parameters

(d) f_{cL30}, best performing parameters

Figure 6.2: Average NFE required to find the i:th optimum. The numbers state what percentage of the runs were able to find the corresponding number of optima, if not 100%. Note the different scales.

good performance can be explained by the ability of DEGS to exploit the regularity of the functions described in Section 5.5. Because global mutation uses unscaled differentials, DELG and DELL are similarly able to exploit the regularity. This enables the three algorithms to locate all the optima using small population sizes, even in functions with a lot of local optima.

In principle, CRDE is able to exploit the regularity as DEGS or DELS-based methods similarly. However, the performance results for CRDE are clearly inferior in all regular cases, and it is the worst performer among the tested methods in f_{cL1}. The reason for the poor performance is the dual nature of parameter F. The value $F = 1$ hinders the local search capabilities of CRDE too much to allow the algorithm to converge near the optima fast enough to start exploiting the regularity efficiently. In both cosine functions, CRDE achieves the best performance with $F = 0.5$ and only in f_{Rshu} does it use the value $F = 1$, which allows the most efficient exploitation of regularity. In cosine functions, where the global optima are in a 5 by 5 grid (always 5 optima in a direct line), the value $F = 0.5$ still allows the algorithm to exploit the regularity rather efficiently, while considerably speeding up the convergence. In f_{Rshu} the optima form two 3 by 3 grids (only 3 optima in a direct line). Now, the value $F = 0.5$ will only allow direct jumps from one optimum to another with differentials generated between endpoints of a line compared to the value of $F = 1$, when any differential between two optima is a potential jump to a new optima. $F = 1$ also allows jumps between optima in different grids, unlike $F = 0.5$. As can be seen in Figure 6.2(c), with $F = 1$ the initial convergence of CRDE is very slow, but once it locates the first few optima, it does not take long to locate the rest.

The performance of DECG is interesting. While it locates the first few optima fast, it fails to exploit the regularity as efficiently as DEGS, DELG and DELL, and is thus considerably slower in locating the rest of the optima. The explanation lies in the crowding process, which keeps the population members diverse by only replacing the closest member. This keeps the population diverse effectively, as individuals already close to an optimum tend to draw the comparison to themselves, shielding inferior individuals located further away. Thus the inferior solutions converge towards the optima only through small steps. This means that crowding is able to keep the population diverse longer, compared to local selection, which allows longer jumps towards the optima. The increased diversity also means that less of the generated differentials are suitable for exploiting the regularity. The speed advantage of DECG is still clear when compared to the CRDE, as the ability to do efficient local search is no longer tied to the ability to exploit regularity.

The deterministic nature of GRGD makes it a difficult method to compare using the cosine family functions. The chosen method of generating the grid makes the algorithm inefficient in searching near the constraints where the majority of the global optima are located. On the other hand, the algorithm will always instantly locate the first nine global optima of f_{cL1}, f_{cL10} and f_{cL30}, because the first nine starting points will be generated directly at the optima. Generating starting points on the constraints at the beginning would allow the algorithm to instantly locate all global optima of these functions, but such a strategy would strongly bias the search to the constraints in a more general case. For this reason, the results for GRGD are more interesting for the quadratic family functions, where they demonstrate the effect of an even coverage of the search space, but are reported for all functions because of completeness.

The performance of RSGD and SDE, which largely neglect the global phase, compares poorly to the other methods on regular functions. RSGD is still among the best performing methods on f_{cL1}, but the performance drops dramatically when local optima are added, making RSGD the worst performer in other regular functions. For a method like RSGD that depends solely on the starting points, it is crucial to have a random point placed in the area of attraction (AOA) of each global optimum. When no local optima exist, the whole search space belongs to the AOA of some global optima. Also each global optimum has a reasonably large area of attraction in f_{cL1} (due the constraints in f_{cL1}, the four corner optima each have AOA of $1/64$ of the whole search space, while the twelve on the edges have $2/64$, and the nine in the middle have $4/64$ each for initial point placement). So, it is rather easy to locate the AOA of each global optimum by random points, and the efficient local search of RSGD can then locate the optima quickly. The addition of local optima reduces the AOA of global optima and makes it more difficult to find them by random point placement.

SDE demonstrates a similar, but less dramatic decrease in relative performance (the difference in performance compared to the other tested methods), as the NLO increase. It is notable, that the value of $\sigma_{rad} = 0.05$ works best in f_{cL1}, but $\sigma_{rad} = 0.2$ is better in f_{cL10} and f_{cL30}. The Euclidean distance between the global optima is 0.2 in these functions. In f_{cL1}, local search is a viable strategy, and thus a rather small value for σ_{rad} works well. The value $\sigma_{rad} = 0.2$ with $F = 1$ is just large enough to allow five neighboring optima inside a single niche in an cross formation, so that two direct lines containing three regular optima points exist. It is now possible to get long enough differentials inside a niche for jumping from a neighboring optimum to another one. This allows SDE to exploit the regularity to some degree and increases performance in cases containing local optima.

Thus, using σ_{rad} values large enough to allow several global optima inside a single niche in regular functions can sometimes be advantageous. However, the regularities are typically only visible in the global scale. Dividing the search space into a smaller niches prevents SDE from recognizing the regularities and finding more than one of the optima inside each niche. As displayed in figure 6.2(d), SDE achieves best performance in f_{Rshu} with the value $\sigma_{rad} = 0.5$ and is able to locate all 18 global optima in all 100 runs. The minimum Euclidean distance between the global optima in f_{Rshu} is approximately 0.88. When SDE is run on the function with an otherwise similar setup, but using $\sigma_{rad} = 2$ (figure not displayed), it is able to locate all optima only in 22% of the runs (and would likely lose some of the optima also in those cases, if the runs were continued long enough). SDE requires considerably larger population sizes in the regular functions compared to the other evolutionary methods to compensate for the weaker global phase.

Figure 6.2(b) demonstrates a dramatic decrease in the performance of DEGS when $F = 0.9$ is used. Now the algorithm is not able to exploit the regularity, and as a result it is able to locate only a few of the optima. In addition, it will lose the already found optima in time and eventually converge to only one. Using a similar setup with CRDE displays a far less dramatic performance decrease, demonstrating the ability of crowding to prevent the population from converging to a single optimum, which happens in DEGS. Using a larger population size would likely allow CRDE to achieve 100% SR in locating all the optima also with $F = 0.9$. However, increasing the population size decreases the convergence speed of CRDE rapidly compared to the other tested algorithms.

Increasing the σ in DELS-based methods also causes a performance drop, because the direct jumps from an optimum to another are no longer of exactly right length and considerably larger population sizes are required to locate all optima reliably. As can be seen in Figure 6.2(b), AJIT suffers the biggest performance drop on f_{cL1}, when $\sigma = 0.1$ is used followed by DITH. The performance drop for AJIG and DITG is small, because the increase in population does not slow down the hybrid as much, due to the efficient local search. In functions with local optima, the effect of increasing σ is similar, and the performance of DELL methods drops much faster compared to AJIG and DITG. The better performance of the dither global mutation compared to additive jitter is expected, because scaling only the mutation step length disrupts the differentials between the optima less than changing the direction as well. The effect of parameter PX is rather small, as long as the extremes are not used. In functions with local optima, using

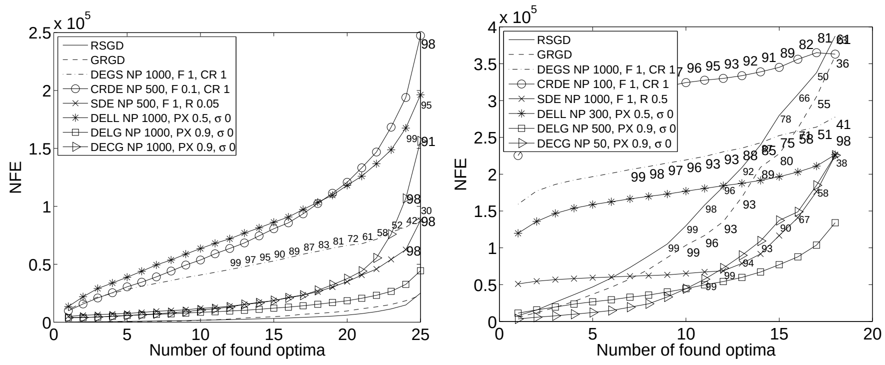
$PX < 0.5$ and $\sigma > 0$ tends to emphasize the local phase too heavily, especially with DELG.

The comparison of SDE performance in Figures 6.2(a) and 6.2(b) shows that decreasing the value of σ_{rad} (or alternatively F) will only speed up the local search to a certain point. Decreasing it further makes the maximum step size too small and starts to slow down the search. A small σ_{rad} may also cause SDE to lose some of the niches due to the selection, which only preserves NP fittest members among the population: if all the niches do not converge at equal speed, it is possible that up to m new points are generated around each seed. In an extreme case, where each seed alone forms a niche, the maximum population could grow to $m \cdot NP$, among which only NP fittest are preserved at the end of the generation. Although typically the number of new points is considerably smaller, it increases when the value of σ_{rad} decreases, and it is a possibility that a niche gets completely erased before it has enough time to converge fully.

Partially regular functions

Table 6.2 and Figure 6.3 show the performance of the algorithms in partially regular functions. As can be seen, DELG achieves the best overall performance. It is the top performer in all partially regular functions containing local optima. However, in f_{Rvin} RSGD is able to outperform DELG and in f_{ScL1} both GD methods demonstrate superior performance to DELG. Both functions contain only global optima. It is notable that the performance of RSGD drops only slightly when f_{cL1} is stretched. The reason is that on average the effect of random stretching on the AOA of optima is quite small, and as RSGD does not use the regularity information, the loss does not affect its performance either. The differences in the AOAs of different optima in f_{Rvin} are considerably larger. This can be clearly seen on the performance, as RSGD can no longer achieve PR 1 in f_{Rvin}, although it is still able to beat the other methods. GRGD outperforms RSGD slightly in f_{SRshu}, but has again problems on the cosine family functions, which have global optima on the constraints. Overall, both GD methods demonstrate rather similar performance, and similarly to the regular functions, the performance of both methods slumps as the NLO increases.

DELL performs poorly in cases with no local optima, but as the number increases, the relative performance improves rapidly and DELL gains the second place in the performance comparison in f_{ScL30} and f_{SRshu}. All DELS-based methods are able to exploit efficiently even the partial regularities with $\sigma = 0$, which gives them an advantage in cases containing local optima. In the functions that contain no local optima, $\sigma > 0$ often performs slightly better, although the differences to results acquired with $\sigma = 0$ are small. On the other hand, increasing the value of σ on functions containing local optima, has typically a notable negative effect on performance. Thus, the use of $\sigma = 0$ is recommended also in partially regular functions. As with the regular functions, the disruptive effect of dither global mutation on the ability of the algorithm to exploit regularity is smaller than that of an additive jitter, and thus increasing σ typically has a slightly smaller negative effect on the performance of DITH and DITG compared to AJIT and AJIG. Using small PX for DELG tends to emphasize the local search too heavily, decreasing the performance in cases with local optima, and $PX = 0.9$ is clearly superior to $PX = 0.5$ in all but f_{ScL1} and f_{Rvin}.



(a) f_{ScL1}, best performing parameters (b) f_{SRshu}, best performing parameters

Figure 6.3: Average NFE required to find the i:th optimum. The numbers state what percentage of the runs were able to find the corresponding number of optima, if not 100%.

Because SDE is unable to exploit the regularity with small σ_{rad} values, the relative performance drop caused by stretching is smaller compared to the other evolutionary methods. This shows especially in the f_{SRshu} and f_{ScL10} functions, where SDE is able to claim the second place in performance comparison. However, in f_{Rvin} SDE is the third worst performer and in the cosine functions, the relative performance of SDE drops with high NLO. The best setup for SDE in all stretched cosine functions is $\sigma_{rad} = 0.05$. While in f_{cL30}, $\sigma_{rad} = 0.2$ is large enough to allow partial exploitation of the regularity, SDE is unable to do that any longer in the stretched function, which explains the significant performance drop in f_{ScL30}.

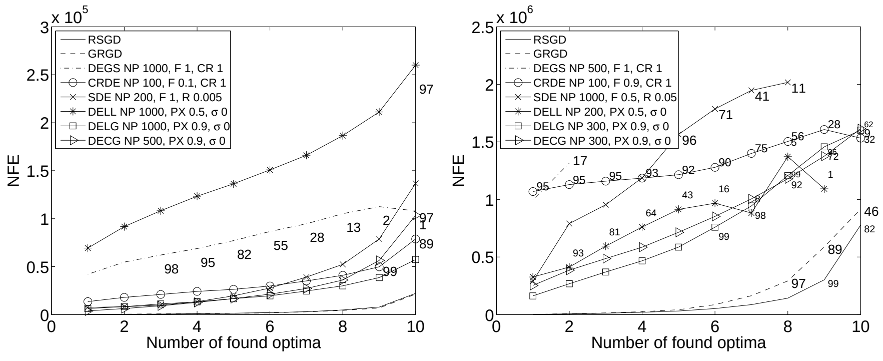
The relative performance of CRDE increases slightly when moving from regular to partially regular functions, but it remains one of the worst performing methods overall, failing to achieve SR of 100% in any of the tested cases. The cosine functions with local optima are especially difficult, and CRDE is the only method which never locates all the optima in f_{ScL10}. It is notable that in functions with no local optima, the best result is achieved with $F = 0.1$, while in cases containing local optima, $F = 1$ is always the best choice in contrast to the regular cases, where $F = 0.5$ performs best in cosine functions. The explanation is again the dual nature of parameter F: in problems with no local optima, a small value of F combined with large *NP* allows a more efficient local search, which outweighs the benefit of exploiting the regularity partially. However, as the NLO increases, relying on the local search becomes inefficient, and the benefit of even partial exploitation of regularity outweighs the loss in local search speed. The ability to exploit the regularity with $F = 0.5$ is diminished, because the number of exploitable differentials is smaller.

DECG is only average performer in the partially regular functions and is the second to worst in f_{ScL1} and f_{Rvin}, which do not contain local optima. It is interesting that in these functions CRDE is able to outperform DECG. The explanation lies in the small

$F = 0.1$ allowed for CRDE, which suits well in functions with no local optima. The relative performance of DECG increases along with NLO similarly as with DELL, but DECG is not able to outperform DELL or DELG in any of the problems. The superior ability to exploit regularities still remains the deciding factor even in partially regular functions. DECG, however, clearly outperforms CRDE in f_{ScL10} and f_{ScL30} but somewhat surprisingly shows inferior performance in f_{SRshu}. The difference there cannot be explained by differences in mutation step length, as CRDE achieves the best result with $F = 1$. Another interesting point to notice is that the best result of DECG on f_{SRshu} is attained with a very small population size, $NP = 50$. This is the only time in the whole test setup that such a small NP value produces the best PR for any method. The explanation probably lies in the very closely situated double global optima of the function. Once the algorithm manages to place one individual in the AOA of one of the double optima, crowding might well use it for comparing any trial generated to the other, as it is very close, and thus either the trial is discarded or the individual jumps from one optimum to the other. It may take a while for the other population members to move close enough to allow the algorithm to locate both optima when the ability to exploit regularity is diminished by stretching the function. Smaller population moves faster as a whole, which could explain why such an small population size works well, but increases the risk that some of the optima are not located at all. Similar effect happens of course with CRDE, but the smoother progress made by the algorithm seems to work somewhat better when the global optima are situated in closely positioned groups.

While DEGS is the top performer in regular functions, the performance crumbles in the partially regular cases. The algorithm is the worst performer in all but the f_{ScL30} function, where it is able to outperform the GD methods. Still, the algorithm is able to exploit even the partial regularities well enough to be able to locate all optima occasionally in all but the f_{Rvin} function. This is quite remarkable, when the fact that it does not use any niching is taken into account. What makes the cosine functions easier for DEGS is the fact that on average the regularity is disrupted less by random stretching compared to f_{Rvin}. Additionally, some of the problem instances will have only slight stretching, which increases the chances for DEGS to jump from the vicinity of an optimum near another and increases the chance to locate multiple optima fast. It is crucial that DEGS is able to locate enough optima fast enough to store the differentials for jumps between optima, before the greedy global selection draws the population to one optimum. The increase in local optima is not as big of a problem for methods able to exploit the regularity, because differentials between local optima are often also suitable for jumping from a local optimum to another and eventually to a global one. In f_{Rvin}, the big differences in AOAs of the optima, combined with the big differences of differentials between different optima (lack of nearly suitable differentials) cause the algorithm to concentrate only on a few optima, typically the ones with large AOA.

When the best performing population sizes are studied in Table 6.3, a substantial increase can be seen for DELS-based methods and especially for DEGS, when moving from regular to partially regular functions, because the methods can no longer rely on regularity as heavily. What may seem odd, is the fact that for most methods, larger NP values are used for the easier f_{ScL1} function compared to f_{ScL10} and f_{ScL30}, which contain local optima. The explanation is simply the NFE_{max}, which limits the usable population size. The methods with an inefficient local phase are forced to use small populations to



(a) $f_{q2,0}$, best performing parameters (b) $f_{q10,0}$, best performing parameters

Figure 6.4: Average NFE required to find the i:th optimum. The numbers state what percentage of the runs were able to find the corresponding number of optima, if not 100%.

converge fast enough. As DELG is able to do more efficient local search, it is able to use a wider range of *NP* values reliably before losing the ability to converge within the given NFE_{max}. DECG actually prefers smaller *NP* on f_{SRshu} and f_{ScL30} compared to their unstretched variants. For DEGS, the main problem is rather to keep the algorithm from converging too fast to a single optimum, and large *NP* values are thus best for all cases.

Irregular functions

Looking at the results in Table 6.2 reveals that although DELG is no longer able to exploit the regularity in the irregular cases, it is again the top performer in both two-dimensional cases which contain local optima. DECG is close behind and demonstrates comparable performance in $f_{q2,1000}$. As the dimensionality increases, RSGD claims the title of top performer in all five and ten-dimensional functions. As can be seen in Figure 6.4(a), the GD methods also outperform other methods in the $f_{q2,0}$ function and are among the top performers in $f_{q2,100}$.

The quadratic family does not offer a meaningful global structure, nor does it offer other easily exploitable features. Furthermore, when different random seeds are used, each quadratic function test set actually contains 100 different functions. This removes exploitable global information effectively. Lobo and Lima [89] claim that multistart methods will very likely outperform EAs in problems with no global structure which the EA could exploit. Although their study only assumes a goal of finding a single optimum, the results in this study suggest that the same generally holds for multimodal optimization and includes the hybrids. The DE part in the hybrids becomes an overhead in irregular functions, as it is no longer able to offer significant advantage to the search through exploiting the global information. Two-dimensional functions containing local optima, however, are an exception for this. As the two-dimensional search space is still rather compact, the niching DE part is able to keep the population well spread, preventing the hybrids from searching the same areas repeatedly. Similarly, GRGD is able to outperform RSGD in $f_{q2,1000}$ through minimizing the redundant searches. As the volume of the search space increases along with the dimensionality, it becomes increasingly difficult to cover the search space with enough detail, and the overhead outweighs the gained benefits, giving an advantage to RSGD over the hybrids. The unbiased sampling of RSGD seems able to offer a better starting point distribution for the higher dimensional quadratic functions compared to GRGD, as the assumption of equally distributed global optima made by the grid fits poorly to the functions. The performance differences between RSGD, GRGD and the hybrids are still rather small in $f_{q5,0}$ and $f_{q10,0}$ which do not contain local optima. When the local optima are added in $f_{q5,100}$ and $f_{q10,100}$, the performance of RSGD drops less compared to the other methods.

DECG outperforms DELG in all five and ten-dimensional functions and is able to challenge GRGD for the second place in performance comparison. While the improved ability of crowding to keep population diversity disrupts the ability to exploit regularities, it is able to offer an advantage over local selection in irregular cases, especially with increasing dimensionality. Both hybrid methods are able to outperform the pure DE-based methods constantly through their more efficient local search capability.

The relative performance of SDE drops again as the NLO increases. The effect of increasing dimensionality is mainly similar. In the cosine functions, where the optima were rather evenly spaced in the search space, the performance of SDE dropped much slower compared to that of RSGD, as the NLO increased. The situation reverses in the two-dimensional quadratic functions, where the optima distribution is random. Of course, the NLO in quadratic cases is significantly lower, so increasing the number further might eventually change the situation. Still, finding a good value for parameter σ_{rad} is more difficult for quadratic functions, because some optima may be very close to each other and the distribution changes each run. Optimally, the value of σ_{rad} should be large enough to allow effective exploration, but at the same time small enough not to place several optima inside the same niche. A good value is easy to define when the optima are spread evenly on the search space, like in the cosine functions, but gets harder if the optima are clustered. This gives a relative advantage to RSGD, in which the distribution of optima affects only through the differences in AOA.

The relative performance of DELL is again poor on the functions with no local optima, but when local optima are added, the relative performance improves slightly. Compared to CRDE, DELL is again able to use larger *NP* to its advantage. As can be seen in the Figure 6.4(b), 5% of the CRDE runs fail to locate any of the optima before reaching NFE_{max} in a function which contains no local optima. Tests with other setups demonstrate that smaller F values enable CRDE to locate the first optima reliably in $f_{q10,0}$, but at a cost of decreased performance in locating the harder optima. Despite the fact that $F = 1$ no longer enables the exploitation of regularity, the explorative power of large F is needed, because CRDE is limited to a rather low *NP*. Thus, the dual nature of F is a problem also in the irregular functions. Although CRDE performs poorly in most irregular functions, it performs relatively well in $f_{q5,0}$ and $f_{q10,0}$. In $f_{q10,0}$ CRDE is the only pure DE method that can even sometimes locate all ten global optima. In the harder functions the algorithm simply converges too slowly to achieve reasonable performance in the given time frame.

Because DEGS relies most on the regularity, its performance drops most, and DEGS is clearly the worst performing method in all irregular functions. In the two-dimensional functions the algorithm is still able to occasionally locate all ten optima, which is actually a notable achievement for an algorithm not using niching, but the performance drops dramatically as the dimension increases. Especially in the ten-dimensional functions, DEGS manages to occasionally locate two of the optima at best. The results demonstrate clearly the inability of DEGS to locate and maintain several optima in functions where the algorithm cannot exploit regularity.

As the ability of DELS-based methods to exploit regularity relies on global mutation, values of $\sigma > 0$ are harmful in functions, that have regularities to exploit. Similarly, a rather large PX value should be preferred for DELG to allow efficient exploitation. However, in the irregular functions, no regularities exist which could be exploited. Using $\sigma > 0$ or smaller values for PX might thus boost the performance.

Table 6.4 demonstrates the effects of using the ajitter and dither versions of global mutation with varying σ values for DELL. As can be seen, using $\sigma > 0$ is able to offer statistically significant improvement to PR only once, in $f_{q10,0}$ using AJIT with large $\sigma = 0.2$. The use of large values for σ tends to decrease the performance more often than offer an advantage, especially on functions containing local optima. The differences between the ajitter and dither versions are small. It seems that typically $\sigma > 0$ is harmful for DELL methods even in completely irregular functions. Often increasings σ allows the algorithm to increase the performance with small population sizes, but a similar result can usually be acquired by using $\sigma = 0$ and increasing the NP instead.

Table 6.5 demonstrates the effects of varying PX and σ for the DELG methods. Using $\sigma = 0.01$ has a rather limited effect on the performance, but is able to give a statistically significant boost in the $f_{q10,100}$ function with the AJIG version. Again, the differences between ajitter and dither are small. Decreasing PX from 0.9 to 0.5, however, has a more notable effect. $PX = 0.5$ is worse on the two-dimensional functions which contain local optima, but offers a statistically significant boost in performance in $f_{q5,100}$ and both ten-dimensional functions. The advantage gained through smaller PX values with the harder functions can be explained by the fact that it decreases the overhead caused by the DE part and takes the algorithm closer to RSGD.

6.1.3 Discussion

Figure 6.5 presents a qualitative summary of the results. As CRDE and SDE are not able to achieve top performance in any of the tested cases they are not included in the figure. DEGS is the best method in all regular cases, but with only a slight margin. For this reason, the graph considers DELG as an equally good choice. Also DELL becomes competitive in regular functions when NLO increases. Outside the regular functions, DEGS and DELL are unable to achieve top performance, and thus no group of functions exist where they would clearly excel over the other compared methods. Although no tests were done on many dimensional regular or partially regular functions, it is expected that the methods able to exploit the regularity in two-dimensional cases efficiently gain even stronger advantage when the dimensionality increases, given an equal degree of regularity.

The GD methods excel in all functions with no local optima. The relative performance, however, decreases rapidly along the increasing number of local optima in regular and

Table 6.4: PR values for varying the σ parameter for the DELL methods using NP^* from Table 6.3 and $PX = 0.5$. All results are compared to DELL with $\sigma = 0$ using the two-tailed Wilcoxon signed-rank test with significance level 0.05. Any results having statistically significant difference are marked with the **bold font** if they are better and with the *italic font* if they are worse.

Function	DELL $\sigma = 0$	AJIT $\sigma = 0.01$	DITH $\sigma = 0.01$	AJIT $\sigma = 0.2$	DITH $\sigma = 0.2$
$f_{q2,0}$	0.997	0.998	1.000	0.999	0.997
std	0.017	0.014	0.000	0.010	0.017
$f_{q2,100}$	0.987	0.986	0.991	0.974	0.984
std	0.039	0.035	0.029	0.061	0.040
$f_{q2,1000}$	0.983	*0.946*	*0.962*	*0.727*	*0.754*
std	0.038	0.083	0.063	0.171	0.165
$f_{q5,0}$	0.777	0.774	0.775	0.789	0.774
std	0.136	0.157	0.151	0.136	0.148
$f_{q5,100}$	0.502	0.477	0.471	*0.044*	*0.087*
std	0.161	0.192	0.176	0.128	0.177
$f_{q10,0}$	0.411	0.391	*0.381*	**0.450**	0.399
std	0.175	0.179	0.170	0.200	0.191
$f_{q10,100}$	0.252	0.252	0.272	*0.133*	*0.129*
std	0.148	0.157	0.162	0.174	0.168

partially regular functions, due to the inability to use global information. DELG excels in all but the easiest of the tested regular and partially regular functions, as well as in the two-dimensional irregular functions which contain local optima. The GD methods perform well overall in the irregular functions. Especially RSGD is the top performer in all five- and ten-dimensional irregular functions. DECG can not match the ability of DELG in exploiting regularity, but performs well in the irregular functions. While not a match for RSGD, DECG outperforms DELG and is able to rival GRGD in the five- and ten-dimensional irregular functions. DECG is also able to rival DELG in two-dimensional irregular functions with high NLO.

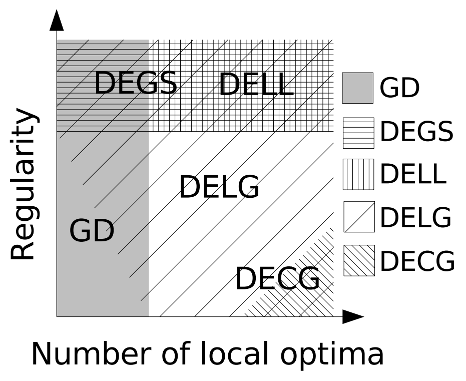
It seems that adding noise to the global mutation is not crucial to the performance, and is in fact harmful unless the function is completely irregular, in which case the noise may be slightly beneficial. Clearly separating the local and global phase is crucial, however, as can be observed by comparing the results of different methods. CRDE excels at keeping the population diversity, but fails to execute efficient local search: the self-adaptation of DE relies on the fast convergence of the population, which decreases the differentials and increases the speed of final convergence. Such convergence does not happen in CRDE. Thus, the user of CRDE is left with an unpleasant choice to either using a large F, which allows effective global search but hampers the local phase, or using a small F, which allows faster local search but hampers the explorative capabilities. The idea to separate

Table 6.5: PR values for varying the σ and PX parameters for the DELG methods using NP^* from Table 6.3. All results are compared to DELG with $\sigma = 0$ and $PX = 0.9$ using the two-tailed Wilcoxon signed-rank test with significance level 0.05. Statistically significantly better results are marked with the **bold font** and significantly worse with the *italic font*. Empty places mean that the particular combination was not tested.

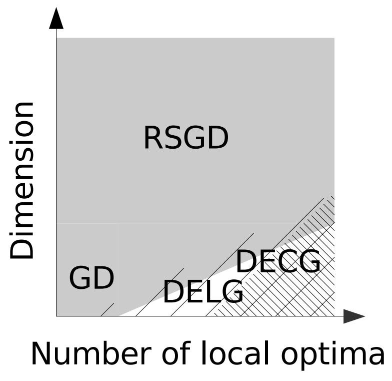
Function	DELG	DELG	AJIG	DITG	AJIG	DITG
	$\sigma = 0$	$\sigma = 0$	$\sigma = 0.01$	$\sigma = 0.01$	$\sigma = 0.01$	$\sigma = 0.01$
	$PX = 0.9$	$PX = 0.5$	$PX = 0.9$	$PX = 0.9$	$PX = 0.5$	$PX = 0.5$
$f_{q2,0}$	1.000	1.000	1.000	1.000		
std	0.000	0.000	0.000	0.000		
$f_{q2,100}$	1.000	*0.990*	0.999	0.996		
std	0.000	0.030	0.014	0.020		
$f_{q2,1000}$	0.990	*0.946*	0.991	0.994		
std	0.030	0.068	0.032	0.030		
$f_{q5,0}$	0.957	0.963	0.959	0.952	0.968	0.970
std	0.062	0.065	0.065	0.067	0.058	0.060
$f_{q5,100}$	0.701	**0.745**	0.710	0.703	**0.745**	**0.750**
std	0.145	0.148	0.160	0.142	0.121	0.140
$f_{q10,0}$	0.893	**0.932**	0.886	0.894	**0.932**	**0.923**
std	0.101	0.075	0.102	0.104	0.083	0.090
$f_{q10,100}$	0.550	**0.619**	**0.584**	0.577	**0.615**	**0.615**
std	0.142	0.150	0.155	0.156	0.153	0.147

the mutation in the local and global part used in DELL allows different values of F to be used and removes the cumbersome dual nature of the parameter. Although this is already beneficial, replacing the local mutation by a more efficient local search method significantly speeds up the local convergence without sacrificing the ability to exploit global information, as demonstrated by the results of hybrids.

As DEGS is not designed for multimodal optimization, the poor results in irregular functions are not surprising. However, the very good performance in regular cases is interesting. The effect is not limited to cases which have multiple global optima, but the algorithm can also use differentials between regularly spaced local optima to locate more optima. This is very important to remember, when constructing test setups for DE based approaches and analyzing the results even in functions with only one global optimum. The ability to exploit regularity is a heritage of using the differentials in the mutation, and the DE-based approaches are hard to beat in regular problems, when a suitable parameter setup is used.



(a) Two-dimensional functions



(b) Irregular functions

Figure 6.5: Best performing methods in different types of problems.

6.2 Locating and maintaining local optima

The purpose of the second test set is to highlight the differences of the compared niching methods in locating and preserving local optima in addition to the global ones. Only the most interesting algorithms from the first test set are included to keep the comparisons manageable, including both hybrids and the CRDE algorithm for comparison. The algorithms are always run for $D * 75000$ function evaluations, after which their average PR is calculated (which is a similar value as used in [163]). For all tests, the value on $NP = 100$ is used. For DELG and DECG, fixed $PX = 0.9$ and $\sigma = 0$ are always used. Some tests were also done using $\sigma > 0$, but they did not produce improved results. For CRDE, $CR = 1$ is always used and for F, values 1 and 0.5 are tested.

The hill-valley function was introduced in [169] to determine if two points in a search space exist in the AOA of same optima. The idea is to simply sample points in a straight line between the two points and to compare the fitness function values of the sampled points with the original points. If any of the interior points have a worse fitness value than either of the original points, the points belong to the AOA of different optima. Thomsen [163] and Zaharie [183] mention the ability of the hill-valley function to prevent CRDE from losing the local optima on the Six-hump camel back function. For this reason, versions of CRDE and DECG using the hill-valley rule are implemented (HCRDE and HDECG) to enable comparison. Five points are always used, for which the first is set in the middle of the line between the original points, and the rest are placed so that they always divide the remaining distance between the previous point and the worse of the original points to half. The hill-valley test is performed in the selection phase, such that a trial is only accepted to the population if it belongs to the AOA of the same optimum. Essentially, the use of the hill-valley function is a way of limiting the global search cababilities of the algorithms in order to decrease REs. A similar effect can be achieved by decreasing the differentials with a small F or using a small PX to emphasize GD. The use of small values for these parameters is also studied along the hill-valley versions.

6.2.1 Function setup

The test set includes all eight common family functions with a single global optimum and the six-hump camel back function. Stretched versions of the six-hump camel back, Shekel's foxholes and both versions of ripple function are also included. Additionally, eight functions from the quadratic and four from the hump families are included to test the performance in cases which contain multiple global optima in addition to the local ones. For both families, the optimum value ranges of local optima of $[-0.5, -0.15]$ and $[-0.95, -0.9]$ are used to demonstrate the effect of having local optima values closer or further away from the global optimum value. For the quadratic functions, two different shapes are used for all optima: rotated ellipsoidal, with values used to generate C from range $[0.003, 0.03]$, and spherical, with a fixed size of 0.02. Half of the quadratic functions are ten-dimensional, to demonstrate the effects of increasing dimension. In all cases, the minimum possible Euclidean distance between two global optima points is set to 0.01. For the hump family, the radii of the optima are either fixed to 0.1 or change in the range $[0.05, 0.2]$. For α the range $[0.2, 0.5]$ is used in all cases. Table 6.6 summarizes the second test function setup.

Table 6.6: The second test set. Global and local columns list the number of global and local optima. The * in the NLO column indicates that the function actually has more local optima, and the value then describes the number of local optima which are defined as interesting to be found. $f(\vec{x}^*)$ is the global optimum value and LOVR is the local optima value range.

Function	NGO	NLO	D	Family	Regularity	$f(\vec{x}^*)$	LOVR	Comment
f_{Rshcb}	2	4	2	common	partial	-1.031	$[0.215, 2.104]$	
f_{SRshcb}	2	4	2	common	irregular	-1.031	$[0.215, 2.104]$	stretched
f_{Rboh}	1	8*	2	common	regular	0	$[0.413, 0.883]$	
f_{Rshe}	1	24	2	common	regular	-499	$[-498, -476]$	
f_{SRshe}	1	24	2	common	partial	-499	$[-498, -476]$	stretched
f_{Rur1}	1	1	2	common	irregular	-6.199	-3.057	
f_{Rur3}	1	4	2	common	regular	-2.5	$[-1.601, -0.700]$	
f_{Rur4}	1	4	2	common	regular	-1.5	-0.621	
f_{Rurw}	1	9	2	common	irregular	-7.307	$[-7.304, -2.931]$	
f_{Rrip}	1	24*	2	common	regular	-2.2	$[-2.109, -0.550]$	
f_{SRrip}	1	24*	2	common	partial	-2.2	$[-2.109, -0.550]$	stretched
f_{Rr25}	1	24	2	common	regular	-2	$[-1.917, -0.502]$	
f_{SRr25}	1	24	2	common	partial	-2	$[-1.917, -0.502]$	stretched
f_{hfl}	10	10	2	hump	irregular	-1	$[-0.95, -0.9]$	fixed radii
f_{hfh}	10	10	2	hump	irregular	-1	$[-0.5, -0.15]$	fixed radii
f_{hcl}	10	10	2	hump	irregular	-1	$[-0.95, -0.9]$	changing radii
f_{hch}	10	10	2	hump	irregular	-1	$[-0.5, -0.15]$	changing radii
f_{qsl2}	10	10	2	quadratic	irregular	-1	$[-0.95, -0.9]$	spherical
f_{qsh2}	10	10	2	quadratic	irregular	-1	$[-0.5, -0.15]$	spherical
f_{qrel2}	10	10	2	quadratic	irregular	-1	$[-0.95, -0.9]$	rot. ellipsoidal
f_{qreh2}	10	10	2	quadratic	irregular	-1	$[-0.5, -0.15]$	rot. ellipsoidal
f_{qsl10}	10	10	10	quadratic	irregular	-1	$[-0.95, -0.9]$	spherical
f_{qsh10}	10	10	10	quadratic	irregular	-1	$[-0.5, -0.15]$	spherical
f_{qrel10}	10	10	10	quadratic	irregular	-1	$[-0.95, -0.9]$	rot. ellipsoidal
f_{qreh10}	10	10	10	quadratic	irregular	-1	$[-0.5, -0.15]$	rot. ellipsoidal

6.2.2 Results and analysis

Table 6.7 presents the peak ratios for the common family functions of the second test set. When looking at the PR for local optima, CRDE with $F = 0.5$ is the top performer in almost all functions. However, in the f_{SRrip} function, it performs very poorly in locating both the global and local optima. To explain this, the properties of the common family functions used in the set must be considered. Because the functions are all two-dimensional, and a majority of them have a low number of optima with reasonably large AOA, a simple local search strategy from the initial population is efficient. Thus smaller F values offer superior performance in most cases. Additionally, in cases where the local search is adequate for locating the optima, using a more effective global search will increase the possibility of losing some of the local optima through making jumps to better areas more likely. Compared to the other functions, f_{Rrip} and f_{SRrip} have a very large number of optima. This is a problem for strategies purely relying on the initial points and local search. However, the value $F = 0.5$ is still suitable to exploit the regularity of the f_{Rrip} function, and CRDE achieves a good performance. When the regularity is disrupted through stretching, a weaker global search capability of CRDE with $F = 0.5$ is revealed, and the peak ratios crumble compared to the results using $F = 1$.

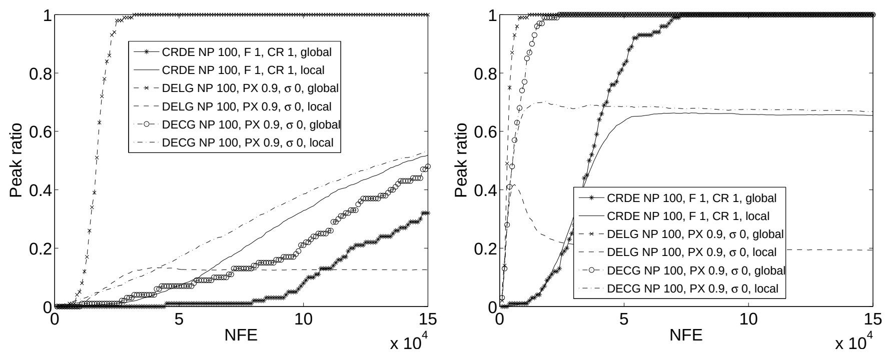
DELG is always able to locate the global optima in all functions, while all methods using crowding have difficulties on f_{SRrip}. On the other hand, DELG has difficulties in keeping the local optima. Excluding the easy f_{Rur1} function, where all methods achieve full peak ratios and six-hump camel back functions, DELG demonstrates clearly inferior performance to all crowding methods. Because DELG has been designed for locating global optima, it always allows jumps which increase the fitness value of the trial compared to its parent. Thus the global optima will eventually draw the population from the local ones. The probability of losing a local optimum is proportional to its optimum value. Better local optima may also draw points from the worse ones, but may only lose points to optima better than they are. Because crowding pits the trial against the closest individual in population, better optima will typically draw less points from the worse. If the other optimum, which the trial is about to jump to, already contains a population member, it is likely that the trial will compete against that instead of allowing a jump from another optimum. Of course, this depends on the function topology, and a steeper optimum may still draw the population away from a shallow one close by, regardless of the crowding. This happens for example in the six-hump camel back functions, where the crowding methods with $F = 1$ are only able to achieve comparable peak ratios to the DELG. The superior ability of DELG to exploit even partial regularities gives it an advantage in locating the global optimum in f_{SRrip}, which has a lot of unwanted local optima to distract the methods.

Figure 6.6(a) displays the development of global and local peak ratios as a function of NFE for f_{SRrip}. As can be seen, DELG achieves global PR of one fast, while the crowding-based methods increase the rate only slowly and are far from one at the 150000 function evaluations. The curves for DECG and CRDE are of similar shape, but CRDE is even slower. DECG and CRDE are able to increase the local PR steadily, while for DELG the rate increases at first, but stops before reaching 0.2 and begins to decrease slowly when DELG starts to lose the worse local optima. Figure 6.6(b) displays the development curves for f_{SRr25}, which serves as a typical example for the curves in most common family functions. Usually DELG is the first to achieve high global PR, but can

Table 6.7: Results for the common family functions of the second test set. The "global" and "local" rows are calculated considering only the number of found global or local optima, while the "all" rows consider both. The **bold font** is used to highlight the best and the *italic font* the worst results compared to others, when the difference to the rest is statistically significant according to the two-tailed Wilcoxon signed-rank test with significance level 0.05.

		DELG	$F = 1$	DECG	$F = 1$	CRDE	$F = 1$	CRDE	$F = 0.5$
Function		PR	std	PR	std	PR	std	PR	std
	Global	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
f_{Rshcb}	Local	0.248	0.025	0.250	0.000	0.250	0.000	**0.488**	0.055
	All	0.498	0.017	0.500	0.000	0.500	0.000	**0.658**	0.037
	Global	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
f_{SRshcb}	Local	0.285	0.087	0.290	0.099	0.275	0.083	**0.460**	0.099
	All	0.523	0.058	0.527	0.066	0.517	0.056	**0.640**	0.066
	Global	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
f_{Rboh}	Local	*0.254*	0.033	1.000	0.000	1.000	0.000	1.000	0.000
	All	*0.337*	0.029	1.000	0.000	1.000	0.000	1.000	0.000
	Global	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
f_{Rshe}	Local	*0.082*	0.007	1.000	0.000	1.000	0.000	1.000	0.000
	All	*0.119*	0.007	1.000	0.000	1.000	0.000	1.000	0.000
	Global	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
f_{SRshe}	Local	*0.240*	0.059	0.968	0.068	0.961	0.076	**0.998**	0.010
	All	*0.270*	0.057	0.969	0.065	0.962	0.072	**0.998**	0.010
	Global	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
f_{Rur1}	Local	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
	All	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
	Global	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
f_{Rur3}	Local	*0.240*	0.049	**0.780**	0.182	0.700	0.170	**0.750**	0.000
	All	*0.392*	0.039	**0.824**	0.146	0.760	0.136	**0.800**	0.000
	Global	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
f_{Rur4}	Local	*0.110*	0.125	0.250	0.000	0.250	0.000	**0.763**	0.152
	All	*0.288*	0.100	0.400	0.000	0.400	0.000	**0.810**	0.122
	Global	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
f_{Rurw}	Local	*0.147*	0.079	0.431	0.060	0.420	0.068	**0.559**	0.019
	All	*0.232*	0.071	0.488	0.054	0.478	0.061	**0.603**	0.017
	Global	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
f_{Rrip}	Local	*0.123*	0.009	0.667	0.000	0.667	0.000	**0.735**	0.030
	All	*0.158*	0.009	0.680	0.000	0.680	0.000	**0.746**	0.029
	Global	**1.000**	0.000	0.480	0.502	0.320	0.469	*0.010*	0.100
f_{SRrip}	Local	*0.125*	0.046	**0.534**	0.122	**0.519**	0.136	0.199	0.086
	All	*0.160*	0.045	**0.532**	0.127	0.511	0.141	0.192	0.082
	Global	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
f_{Rr25}	Local	*0.083*	0.013	0.625	0.000	0.625	0.000	**0.730**	0.026
	All	*0.120*	0.012	0.640	0.000	0.640	0.000	**0.741**	0.025
	Global	1.000	0.000	1.000	0.000	1.000	0.000	0.990	0.100
f_{SRr25}	Local	*0.193*	0.045	0.668	0.098	0.655	0.102	**0.821**	0.122
	All	*0.226*	0.043	0.681	0.093	0.668	0.098	**0.828**	0.117

not keep the local optima well. DECG is a bit behind, and CRDE takes more function evaluations to locate the optima. However, the speed of CRDE and DECG is close to similar in the regular f_{Rshe} and f_{Rrip} functions, where all methods can easily exploit the regularity. Additionally, in the f_{Ruf3} function, which has the optima in a single row, CRDE is faster than DECG in locating the global optimum. The peak curves for CRDE using $F = 0.5$ are comparable to the $F = 1$ curves, except that the rise of global PR in the ripple functions is slower.



(a) f_{SRrip} (b) f_{SRr25}

Figure 6.6: Development of local and global PR.

Effects of limiting global search capability

The results in Table 6.7 demonstrate superior peak ratios for CRDE using $F = 0.5$ instead of $F = 1$ on all but the f_{SRrip} function. Additionally, the hill-valley detection has been shown to improve the ability of CRDE to keep local optima in the six-hump camel back function ([163],[183]). Basically the effect of using the hill-valley function or a small F or PX values is similar: it reduces or even disables the global search capability of the algorithm by preventing jumps between optima. To study this effect further, a set of tests were performed with different versions of the algorithms with limited global search capability using the six-hump camel back function and different ripple functions. The results are displayed in Table 6.8.

As can be seen, limiting the global search capability is advantageous in functions with only a low number of optima. However, for the f_{Rrip} and f_{SRrip} functions, which have a large number of optima and contain regularities, such a limitation is disastrous for the performance. The hill-valley detection is used in a very limiting manner, which completely prevents population points from leaving the AOA of an optimum. Basically the effect is similar to using $PX = 0$, which simply runs gradient descent for each point in the initial population, although the convergence is slower with the hill-valley version. As no jumps are allowed between optima (assuming that the hill-valley is always able to identify a hill between the points, which may not be the case), the algorithms are limited to the optima included in the initial population. In essence such algorithms completely eliminate all replacement errors, not only the UREs. Because the algorithm can never

Table 6.8: Results demonstrating the effects of limiting global search capability. The values are the average PR for all optima (both local and global). The **bold font** is used to highlight the best and the *italic font* the worst results compared to others, when the difference to the rest is statistically significant according to the two-tailed Wilcoxon signed-rank test with significance level 0.05.

	DELG	DECG	CRDE	DELG	DECG	CRDE	DELG	HDECG	HCRDE
Function	F=0.5	F=0.5	F=0.5	F=0.1	F=0.1	F=0.1	PX=0	F=1	F=0.5
f_{Rshcb}	0.667	0.667	0.658	**0.998**	**1.000**	**1.000**	**1.000**	**0.998**	**1.000**
std	0.000	0.000	0.037	0.017	0.000	0.000	0.000	0.017	0.000
f_{Rrip}	0.254	**0.824**	0.746	0.159	0.273	0.149	0.004	0.009	*0.000*
std	0.029	0.060	0.029	0.067	0.099	0.068	0.013	0.018	0.000
f_{SRrip}	0.253	**0.352**	0.192	0.214	0.260	0.139	0.007	0.012	*0.001*
std	0.056	0.098	0.082	0.081	0.095	0.079	0.017	0.023	0.006
f_{Rr25}	*0.126*	0.831	0.741	0.979	0.990	0.984	0.973	0.991	**0.998**
std	0.018	0.053	0.025	0.030	0.018	0.023	0.029	0.018	0.010
f_{SRr25}	*0.274*	0.858	0.828	0.967	**0.986**	0.976	0.953	**0.981**	**0.984**
std	0.081	0.107	0.117	0.036	0.028	0.030	0.046	0.032	0.029

lose any of the found optima, it will be stuck with the first found ones. In cases where *NP* is significantly smaller than the number of optima, it is unlikely that the first found optima are the best ones. Such a severe limitation of the global search capability can offer an advantage in functions where the global search phase is unable to offer advantage to the search through exploiting the global function features. Still, the multistart methods are a superior choice for such functions, because they do not require a predetermined population size and lack the overhead related to the use of population.

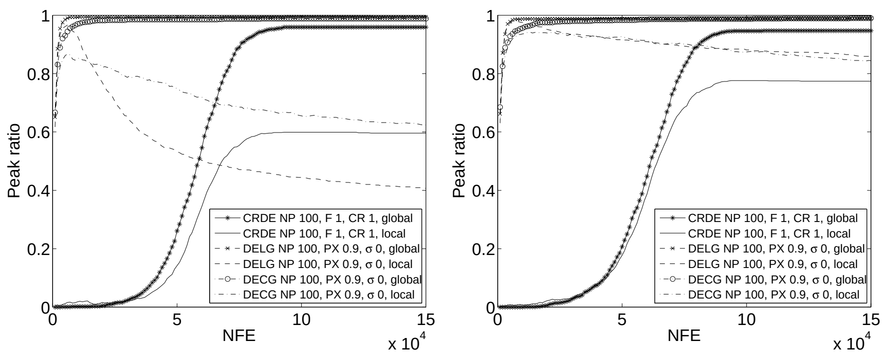
Using value $F = 0.1$ is not as strict a limitation as $PX = 0$ or the hill-valley detection, and allows small jumps to neighboring optima. This shows as an improved performance with the f_{Rrip} and f_{SRrip} functions. Increasing F to 0.5 similarly improves the performance in f_{Rrip} and f_{SRrip}, but decreases the performance in the easier functions. While a small F is not as limiting, it still prevents the algorithm from doing effective global search. The use of a small F means that each population member performs extended local search: it can jump out of small local optima and find a locally global solution, but cannot not exploit global information, like regularities. Thus algorithms using implicit PN will behave like explicit PN methods, if the step length is reduced enough, as they lose the global search phase and their sight becomes limited on a part of the search space.

The tendency of DELG to lose the local optima is still visible in results acquired with value $F = 0.5$, but the difference to crowding methods vanishes when F is decreased further, because no long jumps are possible from worse to better local optima. The results with $F = 0.5$ for DECG demonstrate that the superior local peak rates for CRDE in Table 6.7 are indeed due to the value of F, and using a similar value for DECG typically offers superior performance.

Quadratic and hump family functions

Looking at the results for the two-dimensional quadratic family functions in Table 6.9, a significant drop in local peak ratios of DELG can be seen in all cases where high local optima value range ($[-0.5, -0.15]$) has been used, compared to the cases with the low ($[-0.95, -0.9]$) range. For crowding methods, the drop is significantly smaller, but still visible. When the optimum values of local optima are close to the local ones, the probability for the DELG to lose them decreases, and the results are comparable to the ones acquired by the crowding methods. In the hump family functions, the drop in local peak ratios is even more clear for all algorithms between the functions with low and high local optima value ranges. Still, the local peak ratios of DELG are significantly inferior compared to the other methods in all functions, and drop to zero in the ones with a high optimum value range. The results demonstrate well the tendency of DELG to lose the poor local optima and the increased, although not perfect, ability of crowding to preserve them. CRDE with $F = 0.5$ achieves the top local peak ratios. DELG achieves the top global peak ratios in most hump family functions, while DECG gains the upper hand in the quadratic family functions with rotated ellipsoidal optima shapes.

Figure 6.7 demonstrates the development of global and local PR as a function of NFE for f_{qsh2} and f_{qsl2}. For DELG and DECG, the global and local peak ratios rise rapidly almost to one in both functions. The local ratios start to decrease after peaking, as the algorithms lose the worse local optima. In f_{qsl2}, DELG and DEGS lose the local optima at a similar slow rate. In f_{qsh2}, the rate is increased for both methods, but especially for DELG, which initially achieves an even higher local peak ratio, but loses the optima very rapidly after that. In the harder f_{qreh2} and f_{qrel2} functions, the development curves behave similarly, but do not achieve as high an initial value for the local peak ratios, and DECG is able to keep it almost stable. The development figures for the hump family functions resemble the quadratic family ones. The peak ratio development curves for CRDE using $F = 0.5$ look similar to the $F = 1$ curves, except that the rise of the global peak ratio is typically slower.



(a) f_{qsh2} (b) f_{qsl2}

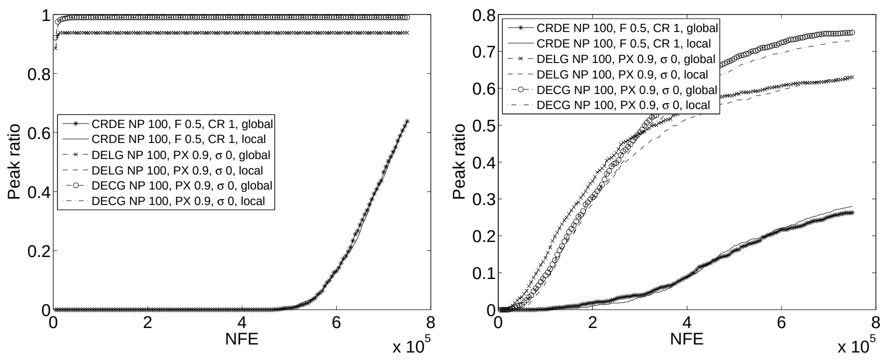
Figure 6.7: Development of local and global PR.

The results for ten-dimensional quadratic functions in Table 6.9 show clearly the advantage of the efficient local search of hybrids. Although the NFE limit is now five times larger compared to the two-dimensional cases, CRDE with value $F = 1$ converges too slowly, and as a result is able to locate almost no optima. Value $F = 0.5$ speeds up the local search and allows the algorithm to locate part of the optima, but the peak

Table 6.9: Results for the hump and quadratic family functions of the second test set. The "global" and "local" rows are calculated considering only the number of found global or local optima, while the "all" rows consider both. The **bold font** is used to highlight the best and the *italic font* the worst results compared to others, when the difference to the rest is statistically significant according to the two-tailed Wilcoxon signed-rank test with significance level 0.05.

		DELG	$F=1$	DECG	$F=1$	CRDE	$F=1$	CRDE	$F=0.5$
Function		PR	std	PR	std	PR	std	PR	std
f_{hfl}	Global	**0.995**	0.022	0.981	0.042	0.969	0.058	0.964	0.085
	Local	*0.310*	0.137	0.551	0.149	0.481	0.149	**0.681**	0.120
	All	*0.653*	0.069	0.766	0.075	0.725	0.085	**0.822**	0.071
f_{hfh}	Global	**0.999**	0.010	0.990	0.033	0.969	0.053	0.981	0.053
	Local	*0.000*	0.000	0.285	0.128	0.261	0.126	**0.382**	0.145
	All	*0.500*	0.005	0.638	0.065	0.615	0.072	**0.682**	0.068
f_{hcl}	Global	0.950	0.070	0.942	0.076	0.944	0.078	0.959	0.067
	Local	*0.381*	0.150	0.599	0.142	0.543	0.145	**0.729**	0.118
	All	*0.666*	0.073	0.771	0.074	0.744	0.083	**0.844**	0.066
f_{hch}	Global	**0.977**	0.047	0.958	0.067	0.958	0.065	**0.984**	0.040
	Local	*0.000*	0.023	0.290	0.173	0.240	0.156	**0.331**	0.172
	All	*0.489*	0.067	0.624	0.091	0.599	0.089	**0.658**	0.089
f_{qsl2}	Global	0.989	0.031	0.990	0.030	*0.947*	0.076	0.992	0.027
	Local	0.859	0.102	0.844	0.113	*0.774*	0.132	**0.900**	0.084
	All	0.924	0.052	0.917	0.056	*0.861*	0.073	**0.946**	0.045
f_{qsh2}	Global	0.994	0.024	0.988	0.036	*0.959*	0.059	0.992	0.031
	Local	*0.407*	0.125	0.624	0.148	0.596	0.148	**0.765**	0.129
	All	*0.701*	0.062	0.806	0.074	0.778	0.072	**0.879**	0.068
f_{qrel2}	Global	*0.887*	0.103	**0.949**	0.066	*0.890*	0.112	0.924	0.090
	Local	0.815	0.128	**0.867**	0.117	0.786	0.127	**0.866**	0.117
	All	0.851	0.084	**0.908**	0.064	0.838	0.084	**0.895**	0.074
f_{qreh2}	Global	0.925	0.078	**0.975**	0.046	0.918	0.090	0.929	0.077
	Local	*0.478*	0.150	0.702	0.138	0.642	0.151	**0.759**	0.136
	All	*0.702*	0.077	**0.839**	0.070	0.780	0.089	**0.844**	0.078
f_{qsl10}	Global	0.922	0.079	**0.996**	0.020	*0.000*	0.000	0.628	0.397
	Local	0.932	0.075	**0.997**	0.017	*0.000*	0.000	0.635	0.385
	All	0.927	0.051	**0.997**	0.013	*0.000*	0.000	0.632	0.386
f_{qsh10}	Global	0.938	0.081	**0.990**	0.030	*0.000*	0.000	0.638	0.377
	Local	0.933	0.081	**0.998**	0.014	*0.000*	0.000	0.637	0.383
	All	0.936	0.059	**0.994**	0.018	*0.000*	0.000	0.638	0.374
f_{qrel10}	Global	0.643	0.151	**0.755**	0.140	*0.000*	0.000	0.278	0.178
	Local	0.621	0.156	**0.738**	0.135	*0.000*	0.000	0.278	0.211
	All	0.628	0.109	**0.747**	0.094	*0.000*	0.000	0.278	0.168
f_{qreh10}	Global	0.630	0.137	**0.751**	0.137	*0.000*	0.000	0.263	0.173
	Local	0.632	0.146	**0.728**	0.130	*0.001*	0.010	0.280	0.204
	All	0.631	0.107	**0.740**	0.090	*0.001*	0.005	0.272	0.161

ratios are still clearly inferior compared to the hybrid methods. This shows well that the lack of efficient local search makes CRDE unusably slow as the dimension of the problem increases, especially with large F values. Figure 6.8 displays the PR development for the f_{qsh10} and f_{qreh10} functions, where the slow convergence of CRDE even with $F = 0.5$ can be seen. All methods are equally good at locating global and local optima, and contrary to the two-dimensional functions, no difference in local PR between the functions with low and high local optima value ranges is visible. As the search space increases along dimensionality, the probability of jumps from local optima to global ones decreases, and allows even DELG to keep the local optima better. The peak ratios for DECG are superior compared to DELG in all ten-dimensional functions, which is in line with the results of the first test set. Using rotated ellipsoidal shapes for optima increase the problem difficulty compared to the spherical ones, which is clearly visible in the PRs of all methods.



(a) f_{qsh10} (b) f_{qreh10}

Figure 6.8: Development of local and global PR.

6.2.3 Discussion

Also the second test set demonstrates the advantage of separating the global and local search operations through hybridization. While CRDE is able to keep up with the hybrids in the easier two-dimensional function,it becomes too slow to achieve convergence in reasonable time when the dimensionality increases. RSGD was not used in the second test set, as the algorithm can not lose local optima and the probability to locate a given optimum is simply proportional to its AOA, regardless of the depth. RSGD would very likely achieve top performance in locating both the global and local optima in most of the used functions (excluding the ripple functions) due to the small NLO and the lack of regularities in the hump and quadratic functions.

Using DECG instead of DELG offers two benefits: crowding is less prone on losing only locally optimal solutions, and the increased population diversity gives an advantage over local selection with irregular functions as the function dimensionality increases. The probability of losing the local optima with local selection, however, also decreases with dimensionality. Still, if the goal is to locate also suboptimal solutions reliably, DECG may be the better choice. If the interest is only in the global or nearly global solutions, DELG becomes more attractive with its superior ability to exploit regularities and the lower computational complexity. DECG is able to offer advantage in irregular functions of higher dimensionality, but as the ability to exploit regularities is the main contribution of the DE part to the hybrid, multistart methods are even more effective in the irregular functions.

The addition of noise to the global mutation by using $\sigma > 0$ did not offer clear advantages in locating the local optima, either. The ability to control the mutation step length through the F parameter, however, is sometimes beneficial also for the global mutation operation. While $F = 1$ generally allows the most effective global search capability, different values may be more suitable for exploiting a particular function topology. Still, good results with a small F are easily misleading, because their usability is limited to cases with only a small number of optima. The same is true for all approaches which limit the global search capability, like the use of the hill-valley function. It would be possible to modify the hill-valley algorithm so that a jump away from an AOA of an optimum is allowed, if at least one other population member is present in the same optimum. However, this would be computationally very expensive, especially because each hill-valley calculation between two points requires several function evaluations. Even if the computational cost is ignored, such a method can still locate only NP optima at best. Because an algorithm using the hill-valley detection can never lose any of the found optima, it will be stuck with the first found ones, and multistart methods can perform the same task more efficiently. In cases where NP is significantly smaller than the number of optima, it is unlikely that the first found optima are the best ones. An approach was also briefly tested, where the trial was allowed to replace the worst population member, in case it was decided to reside in a different optimum than the target to circumvent the limitation of never losing optima. Such an algorithm, however, behaves like global selection and converges fast into a single point, losing the ability to do multimodal optimization.

6.3 Summary

The aim of this section was to increase the understanding of the role of local and global search in multimodal optimization through studying eight different optimization approaches. The results demonstrate the capabilities of Differential Evolution-based algorithms to identify and exploit regularities efficiently, due to use of differentials in mutation. Even without any external niching methods, DE can achieve very good performance in regular functions and works reasonably well as long as even partial regularities remain. Methods having a strong global phase generally outperform the local search-oriented methods in regular functions.

The suggested hybrid approaches combine global search aspect provided by the DE successfully to the efficient local search capability provided by the gradient descent algorithm. The resulting algorithms are able to provide increased performance compared to both parent algorithms, as long as even partial regularities exist, which the DE part can exploit. This implies significant advantages to be attainable in multimodal optimization through separation of the local and global phases of an algorithm and selecting appropriate and effective methods for both. If the function to be optimized does not contain regularities or other features the global phase can exploit, methods concentrating only on the local phase gain advantage, as the global phase then adds only overhead to the process. This is demonstrated by the fact that the multistart methods are the top performers in the irregular functions, which do not offer exploitable global features. The more efficient local phase of the hybrids, however, helps them to achieve clearly superior performance compared to pure DE methods also in irregular functions. The lack of an efficient local phase in pure DE methods often slows down the convergence too much to fully take advantage of the effective global phase, even in functions which contain exploitable features. As a consequence, algorithms having a weak local phase are often forced to use inefficient parameter setups, like too small population sizes or too short mutation steps, to allow convergence in a reasonable time.

The comparison of local selection and crowding revealed that local selection is more apt to losing already found local optima, especially if their fitness differs much from the globally optimal fitness. While local selection demonstrates superior performance with functions containing regularities, crowding gains advantage in irregular functions when the dimensionality increases. Multistart methods, however, generally outperform even DECG in irregular functions, because the global phase is not able to contribute enough to the search to offset the overhead. The true strength of the DE hybrids compared to the multistart methods lies in the fact that they are able to offer a potential increase in performance through identifying and exploiting regularities the user is not aware of beforehand. If such regularities are not available, they still offer superior performance to pure DE methods.

The results also question the advantage of using randomization in the mutation of DE when performing multimodal optimization, as using $\sigma > 0$ did not offer clear advantages in any of the tested setups. The ability to identify and exploit regularities is a major advantage of DE in multimodal optimization, and the randomization directly hinders the algorithm in this respect. The DELG and DECG algorithms could be simplified by removing the randomization from the global mutation. This would reduce the parameters by one, as σ would no longer be needed. An advantage gained from a random element in mutation is often reported for noisy functions, however, and as the test setup did not contain any such functions, it is premature to decide the usefulness of the operator. However, for non-noisy functions, the use of $\sigma = 0$ is recommended, at least as an initial value.

Table 6.10 lists the complete set of parameters for each tested algorithm and classifies them to three groups according to the difficulty of finding a good value for each. The classification is somewhat qualitative. The "easy" control parameters either have a clearly optimal setup, which allows the best use of the strengths a particular algorithm (F, CR, σ, CF) or they are less problem dependent and the algorithm is not sensitive to slight changes of the parameter (PX, δ, η). This allows efficient use of the algorithms by simply using suggested values for these parameters. Parameters classified as "moderate", are problem dependent, but the algorithms are not highly sensitive to these parameters. Mainly the population size is moderately difficult parameter to set for most methods, because using overly large population sizes typically only decreases the efficiency of the algorithms, but is not critical to the success. For methods having especially weak local search capability, the algorithm becomes more sensitive to the NP and thus for CRDE the selection of the parameter is "hard". Also the F is hard to tune for CRDE because of the dual nature of the parameter. SDE is very sensitive to the parameter σ_{rad}, which is highly problem dependent and thus hard to tune. The F and σ_{rad} are related and changing the value of σ_{rad} affects the length of the differentials as well as the value for F. For this reason, the selection of the value for F is dependent on the value of σ_{rad}, which increases the difficulty of selecting a good value for F.

Table 6.10: Complete list of the control parameters of the tested methods classified according to the difficulty of specifying good values for each and recommended values for the less problem dependent parameters.

Method	Easy	Moderate	Difficult
DELG	$F = 1$ $\sigma = 0$ $PX = 0.9$ $\delta = 10^{-3}$ $\eta = 10^{-8}$	NP	
DECG	$F = 1$ $\sigma = 0$ $PX = 0.9$ $CF = NP$ $\delta = 10^{-3}$ $\eta = 10^{-8}$	NP	
CRDE	$CR = 1$ $CF = NP$		NP F
DELL	$F = 1$ $\sigma = 0$ $PX = 0.5$	NP	
SDE	$m = 10$ $CR = 1$	NP F	σ_{rad}
DEGS	$F = 1$ $CR = 1$	NP	
RSGD	$\delta = 10^{-3}$ $\eta = 10^{-8}$		
GRGD	$\delta = 10^{-3}$ $\eta = 10^{-8}$		

Chapter VII

Conclusions

The main objective of this thesis was to increase the understanding of the role of local and global search in multimodal optimization. The results imply a significant advantage to be attained in multimodal optimization through separation of the local and global phases of an algorithm and selecting appropriate and effective methods for both. A natural way of attaining this is by hybridization. The study reveals the ability of hybrid approaches to outperform non-hybrid methods through their more effective use of global and local search phases in functions which contain suitable global features to be exploited. If such features do not exist, the global phase becomes a burden. An efficient local phase, however, is always important, and the lack of one may severely hinder also the ability of an algorithm to exploit the global features.

The most obvious methods concentrating only on the local phase are the multistart methods. Many evolutionary methods, however, also perform extended local search, and largely neglect the global search phase. The use of explicit parallel niching methods or simply reducing the step length enough have similar effects in limiting the global search capabilities of the algorithm. Such methods are efficient in functions with a low amount of optima and ones without suitable global features which could be exploited. The lack of an effective global search phase, however, prevents the methods from identifying and exploiting the global function features, which is the central idea of evolutionary algorithms.

The lack of an efficient local phase slows down the actual convergence and prevents the algorithms from taking full advantage of the effective global phase. Self-adaptive algorithms like Differential Evolution, which use the population information to scale the step length, are especially problematic in the context of multimodal optimization. When the aim is to locate only a single optimum, the algorithm may rely on the converging population to shorten the step length and increase the local search capacity near the global optimum. In multimodal optimization, similar general convergence does not happen, as the population needs to stay spread between several optima. Simply implementing a niching method for a self-adaptive algorithm without considering the effects to the local search capability may thus lead to a very slowly converging algorithm. Such algorithms are often forced to use inefficient parameter setups, such as too small population sizes or too short mutation steps, to allow convergence in reasonable time, which in turn hampers the global search capability.

The idea of separating the global and local search phase can be used in the future to generate different methods for multimodal optimization. Differential Evolution is able to identify and exploit the regularities of the function efficiently through the mutation operation, which is based on differentials between population members. Additionally, the ability to exploit separability can be gained through crossover. Different approaches could be used to identify and exploit different features. Thus a wide variety of different approaches for implementing the global and also the local part should be studied to determine the best combinations for different types of problems. Another potential research direction could be to study the ways and possible benefits of making the *PX* parameter adaptive. The advantage of the hybrid algorithms lies in the fact that they are able to offer a potential increase in performance compared to multistart methods through identifying and exploiting global features the user is not aware of beforehand, but if such features are not available, they still offer superior performance to pure EA methods through the more efficient local search.

The proposed test function framework allows the generation of different types of multimodal test functions, and allows features of the functions to be changed independently of each other. This makes it possible to study how each function feature affects the performance of each algorithm. Especially interesting in the context of this study is the regularity. Interestingly, the basic version of the Differential Evolution algorithm, which does not use any niching methods, demonstrates the best performance in locating the global optima in regular functions. The algorithm is not suitable for multimodal optimization in a more general setup, however, as it quickly converges to a single optimum without the regularities. The result demonstrates the need for a well defined and understood test setup, as the results could otherwise be misleading.

Local selection was found to be able to preserve population diversity, and as a result to be suitable to be used as a niching method. The advantages of the approach are the lack of added control parameters and low computational complexity. The comparison of local selection and crowding revealed that local selection is more apt to losing already found local optima, especially if their fitness differs much from the globally optimal fitness. While local selection demonstrates superior performance in functions containing regularities, crowding gains advantage in irregular functions when the dimensionality increases. The price to pay is the higher computational complexity of crowding.

Based on the acquired results, DELG offers the most potential for new arbitrary multimodal optimization problems through combining the ability to exploit possible regularities to the capability for efficient local search. If also local optima which have considerably inferior fitness are of interest, DECG becomes an interesting choice. If the hybrids demonstrate poor performance, the particular problem likely does not contain regularities which could be exploited and RSGD will then become the algorithm of choice. The pure DE methods can not be recommended due to their inferior local search capability. Also GRGD is an inferior choice to RSGD in arbitrary functions where no priori information is available due the biased starting point distribution.

The use of randomization in the mutation of Differential Evolution hinders the ability of the algorithm to exploit regularities and is thus typically harmful in the context of multimodal optimization. The possible advantages of the randomization was not confirmed by the results in this thesis and remains a future research topic.

Bibliography

[1] Aicholzer, O., Aurenhammer, F., Brandstätter, B., Ebner, T., Krasser, H., and Magele, C. Niching evolution strategy with cluster algorithms. In *Proceedings of the 9th Biennial IEEE Conference on Electromagnetic Field Computations* (2000).

[2] Ando, S., Sakuma, J., and Kobyashi, S. Adaptive isolation model using data clustering for multimodal function optimization. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO 2005)* (Washington DC, USA, 2005), ACM Press, pp. 1417–1424.

[3] Avriel, M. *Nonlinear Programming: Analysis and Methods.* Dover Publishing, 2003. ISBN 0-486-43227-0.

[4] Ballester, P., and Carter, J. An effective real-parameter genetic algorithm with parent centric normal crossover for multimodal optimisation. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2004)*, vol. 3120, Springer, pp. 901–913. ISBN 978-3-540-22344-3.

[5] Ballester, P., and Carter, J. An effective real-parameter genetic algorithm for multimodal optimization. In *Proceedings of Conference on Adaptive Computing in Design and Manufacture (ACDM 2004)* (2004), Springer, pp. 359–364.

[6] Beasley, D., Bull, D., and Martin, R. An overview of genetic algorithms: Part 2, research topics. *University Computing 15* (1993), 170–181.

[7] Beasley, D., Bull, D. R., and Martin, R. R. A sequential niche technique for multimodal function optimization. *Evolutionary Computation 1*, 2 (1993), 101–125.

[8] Bessaou, M., Pétrowski, A., and Siarry, P. Island model cooperating with speciation for multimodal optimization. In *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature* (London, UK, 2000), Springer-Verlag, pp. 437–446.

[9] Beyer, H., and Schwefel, H. Evolution strategies: A comprehensive introduction. *Natural Computing 1*, 1 (2002), 3–52.

[10] Bezier, P. How renault uses numerical control for car body design and tooling. In *SAE Paper 680010, Society of Automotive Engineers Congress* (Detroit, Michigan, USA, 1968).

[11] Bezier, P. *The Mathematical Basis of UNISURF CAD System.* Butterworth-Heinemann, 1986. ISBN 978-0408221757.

[12] Blum, C., and Merkle, D., Eds. *Swarm Intelligence Introduction and Applications.* Springer, 2008. ISBN 978-3-540-74088-9.

[13] Branke, J. Memory-enhanced evolutionary algorithms. In *Proceedings of the Congress on Evolutionary Computation (CEC 1999)* (1999), vol. 3, pp. 1875–1882.

[14] Branke, J. *Evolutionary Optimization in Dynamic Environments.* Kluwer Academic Publishers, Norwell, MA, 2002.

[15] Brits, R., Engelbrecht, A., and van den Bergh, F. A niching particle swarm optimizer. In *Proceedings of 4th Asia-Pacific Conference of Simulated Evolution Learning* (2002), pp. 692–696.

[16] Brits, R., Engelbrecht, A., and van den Bergh, F. Solving systems of unconstrained equations using particle swarm optimization. In *Proceedings of the 2002 IEEE International Conference on Systems, Man and Cybernetics* (6-9 October 2002), vol. 3, pp. 102–107. ISBN: 0-7803-7437-1.

[17] Bäck, T., Fogel, D., and Michalewicz, Z., Eds. *Evolutionary Computation 1: Basic algorithms and Operators.* IOP Publishing, 2000. ISBN 0 7503 0664 5.

[18] Bäck, T., Fogel, D., and Michalewicz, Z., Eds. *Evolutionary Computation 2: Advanced algorithms and Operators.* IOP Publishing, 2000. ISBN 0 7503 0665 3.

[19] Bäck, T., and Schwefel, H. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation 1*, 1 (1993), 1–23.

[20] Caponio, A., Neri, V., and Tirronen, V. Super-fit control adaptation in memetic differential evolution frameworks. *Soft Computing - A Fusion of Foundations, Methodologies and Applications* (2008), 811–831.

[21] Caruana, R., Schaffer, J., and Eshelman, L. Using multiple representations to improve inductive bias: Gray and binary coding for genetic algorithms. In *Proceedings of the sixth international workshop on Machine learning* (San Francisco, USA, 1989), Morgan Kaufmann, pp. 375–378. ISBN 1-55860-036-1.

[22] Cavicchio, D. *Adapting Search Using Simulated Evolution.* PhD thesis, University of Michigan, Ann Arbor, Michigan, 1970.

[23] Chakraborty, U., Ed. *Advances in Differential Evolution*, vol. 143. Springer, 2008. ISBN 978-3-540-68827-3.

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

[25] Chiou, J., and Wang, F. Hybrid method of evolutionary algorithms for static and dynamic optimization problems with application to a fed-batch fermentation process. *Computers and Chemical Engineering 23* (1999), 1277–1291.

[26] Chong, E., and Zak, S. *An Introduction to Optimization*, 2 ed. John Wiley & Sons, 2001.

[27] Cioppa, A., Stefano, C., and Marcelli, A. On the role of population size and niche radius in fitness sharing. *IEEE transactions in evolutiona computation 8*, 6 (December 2004), 580–592.

[28] Crutchley, D. A., and Zwolinski, M. Using evolutionary and hybrid algorithms for dc operating point analysis of nonlinear circuits. In *Proceedings of 2002 IEEE World Congress on Computational Intelligence* (Honolulu, USA, 12-15 May 2002), pp. 753–758. ISBN 0-7803-7282-4.

[29] Cvijovic, D., and Klinowski, J. *Handbook of Global Optimization*, vol. 2. Springer, 2002, ch. Taboo Search: An Approach to the multiple-minima Problem for Continuous functions, pp. 387–406. ISBN 1-40200-632-2.

[30] Darwen, P. J., and Yao, X. A Dilemma for Fitness Sharing with a Scaling Function. In *Proceedings of the Second IEEE International Conference on Evolutionary Computation* (Piscataway, New Jersey, 1995), IEEE Press, pp. 166–171.

[31] Das, S., Konar, A., and Chakraborty, U. Improved differential evolution algorithms for handling noisy optimization problems. In *Proceedings of the Congress on Evolutionary Computation (CEC 2005)* (2005), vol. 2, pp. 1691– 1698. ISBN 0-7803-9363-5.

[32] Das, S., Konar, A., and Chakraborty, U. Two improved diferential evolution schemes for faster global search. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO 2005)* (2005), pp. 991 – 998. ISBN 1-59593-010-8.

[33] De Jong, K. A. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[34] Deb, K. A population-based algorithm-generator for real-parameter optimization. *Soft Computing 9* (2005), 236–253.

[35] Deb, K., Anand, A., and Joshi, D. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation 10*, 4 (2002), 371–395.

[36] Deb, K., and Goldberg, D. An investigation of niche and species formation in genetic function optimization. In *Proceedings of third international conference on genetic algorithms (ICGA-89)* (1989), pp. 42–50.

[37] Deb, K., Sinha, A., and Kukkonen, S. Multi-objective test problems, linkages, and evolutionary methodologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006)* (2006), ACM Press, pp. 1141–1148.

[38] Dong, Z., Lu, M., Lu, Z., and Wong, K. A differential evolution based method for power system planning. In *Proceedings of 2006 IEEE World Congress on Computational Intelligence* (Vancouver, Canada, 16-21 July 2006), pp. 2699–2706. ISBN 0-7803-9489-5.

[39] Eiben, A., and Schippers, C. On evolutionary exploration and exploitation. *Fundamenta Informaticae 35* (1998), 35–50.

[40] Eiben, A., and Smith, J. *Introduction to evolutionary computing.* Springer, 2003.

[41] Ester, M., Kriegel, H., Sander, J., and Xiaowei, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining* (Portland, Oregon, 1996), AAAI Press, pp. 226–231.

[42] Feoktistov, V., and Janaqi, S. Generalization of the strategies in differential evolution. In *Proceedings of the 18th Internal Parallel and Distributed Processing Symposium (IPDPS 2004)* (2004), pp. 165–170. ISBN: 0-7695-2132-0.

[43] Fogel, L. *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming.* Wiley-Interscience, 1999. ISBN 978-0471332503.

[44] Fogel, L., Owens, A., and Walsh, M. *Artificial Intelligence Through a Simulation of Evolution.* John Wiley & Sons, 1966.

[45] Gallagher, M., and Yuan, B. A general-purpose tunable landscape generator. *IEEE Transactions on Evolutionary Computation 10* (2006), 590–603.

[46] Gan, J., and Warwick, K. A genetic algorithm with dynamic niche clustering for multimodal function optimisation. Tech. Rep. 98-001, Cybernetics Dept, Reading University, 1998.

[47] Gao, Y., and Wang, Y. A memetic differential evolutionary algorithm for high dimensional functions' optimization. In *Proceedings of the Third international Conference on Natural Computation ICNC 2007* (Haikou, Hainan, China, 24-27 August 2007), pp. 188–192.

[48] Gaviano, M., Kvasov, D., Lera, D., and Sergeyev, Y. Algorithm 829: Software for generation of classes of test functions with known local and global minima for global optimization. *ACM Transactions on Mathematical Software 29*, 4 (December 2003), 469–480.

[49] Geem, Z., kim, H., and Loganthan, G. A new heuristic optimization algorithm: Harmony search. *Simulation 7*, 2 (2001), 60–68.

[50] Ghosh, A., Tsutsui, S., Tanaka, H., and Corne, D. Genetic algorithms with substitution and re-entry of individuals. *International Journal of Knowledge-Based Intelligent Engineering Systems 4*, 1 (2000), 64–71.

[51] Glover, F. Tabu search - part i. *ORSA Journal on Computing 1*, 3 (1989), 190–206.

[52] Glover, F. Tabu search - part ii. *ORSA Journal on Computing 2*, 1 (1990), 4–32.

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

[54] Goldberg, D., and Deb, K. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1991, pp. 69–93.

[55] Goldberg, D. E., Deb, K., and Horn, J. Massive multimodality, deception, and genetic algorithms. In *Parallel Problem Solving from Nature 2* (Amsterdam, 1992), R. Männer and B. Manderick, Eds., Elsevier Science Publishers, B. V., pp. 37–48.

[56] Goldberg, D. E., and Richardson, J. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms* (Massachusetts, USA, 1987), J. Grefenstette, Ed., pp. 41–49.

[57] Gustafson, S., and Burke, E. The speciating island model: an alternative parallel evolutionary algorithm. *Journal of Parallel Distributed Computing 66*, 8 (2006), 1025–1036.

[58] Hanagandi, V., and Nikolau, M. A hybrid approach to global optimization using a clustering algorithm in genetic search framework. *Computers and Chemical Engineering 22*, 12 (1998), 1913–1925.

[59] Hansen, N., and Ostermeier, A. Completely derandomized self adaptation in evolution strategies. *Evolutionary Computation 9*, 2 (2001), 159–195.

[60] Harik, G. R. Finding multimodal solutions using restricted tournament selection. In *Proceedins of the Sixth International Conference on Genetic Algorithms* (San Francisco, CA, 1995), L. Eshelman, Ed., Morgan Kaufmann, pp. 24–31.

[61] Hart, W., Krasnogor, N., and Smith, J., Eds. *Recent Advances in Memetic Algorithms*. Springer, 2005. ISBN 3-540-22904-3.

[62] Hendershot, Z. A differential evolution algorithm for automatically discovering multiple global optima in multidimensional, discontinuous spaces. In *Proceedings of Fifteenth Midwest Artificial Intelligence and Cognitive Sciences Conference* (Chicago, 16-18 April 2004), pp. 92–97.

[63] Herrera, F., Lozano, M., and Verdegay, J. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review 12*, 4 (1998), 265–319.

[64] Hock, W., and Schittowski, K. *Test Examples for Nonlinear Programming Codes, Lecture Notes in Economics and Mathematical Systems*, vol. 187. Springer, 1981. ISBN 3-540-10561-1.

[65] Holland, J. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[66] Holland, J. *Adaptation in Natural and Artificial Systems*. MIT Press, 1992. ISBN 0-262-08213-6.

[67] Hooke, R., and Jeeves, T. Direct search solution of numerical and statistical problems. *Journal of the ACM 8*, 2 (1961), 212–229.

[68] Hoos, H., and Stützle, T. *Stochastic Local Search Foundations and Applications.* Morgan Kaufmann, 2004.

[69] Im, C., Kim, H., Jung, H., and Choi, K. A novel algorithm for multimodal function optimization based on evolution strategy. *IEEE Transactions on Magnetics 40*, 2 (2004).

[70] Kan, A., and Timmer, G. Stochastic global optimization methods part i: Clustering methods. *Mathematical Programming 39*, 1 (1987).

[71] Kan, A., and Timmer, G. Stochastic global optimization methods part ii: Multi level methods. *Mathematical Programming 39*, 1 (1987).

[72] Kennedy, J. The particle swarm: social adaptation of knowledge. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 1997)* (Indianapolis, USA), pp. 303–308. ISBN: 0-7803-3949-5.

[73] Kennedy, J. Stereotyping: Improving particle swarm performance with cluster analysis. In *Proceedings of the Congress on Evolutionary Computation (CEC 2000)* (2000), pp. 1507–1512.

[74] Kennedy, J., and Ebenhart, R. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks (ICNN 1995)* (Perth, Australia, 1995), pp. 1942–1948.

[75] Kennedy, J., Eberhart, R., and Shi, Y. *Swarm Intelligence.* Morgan Kaufmann, 2001.

[76] Kirkpatrick, S., Gelatt, C., and Vecchi, M. Optimization by simulated annealing. *Science 220* (1983), 671–680.

[77] Lampinen, J. A constraint handling approach for the differential evolution algorithm. In *Proceedings of IEEE World Congress on Computational Intelligence (WCCI 2002)* (Honolulu, Hawaii, USA, 12-17 May 2002), vol. 2, pp. 1468–1473. ISBN 0-7803-7282-4.

[78] Lampinen, J. Multi-constrained nonlinear optimization by the differential evolution algorithm. In *Soft Computing and Industry - Recent Advances.* Springer, 2002, pp. 305–318. ISBN 1-85233-539-4.

[79] Lampinen, J., and Storn, R. *New Optimization Techniques in Engineering.* Springer-Verlag, 2004, ch. Differential Evolution, pp. 123–166. ISBN: 3-540-20167.

[80] Lampinen, J., and Zelinka, I. On stagnation of the differential evolution algorithm. In *Proceedings of 6th MENDEL International Conference on Soft Computing* (Brno, Czech Republic, 7-9 June 2000), pp. 76–83. ISBN 80-214-1609-2.

[81] Lee, C., Cho, D., and Jung, H. Niching genetic algorithm with restricted competition selection for multimodal function optimization. *IEEE Transactions on Magnetics 35*, 3 (1999).

[82] Lee, K., and Geem, Z. A new meta-heuristic algorithm for continuous engineering optimization: Harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering 194* (2005), 3902–3933.

[83] Lee, Y., and Lee, M. A shape-based block layout approach to facility layout problems using hybrid genetic algorithm. *Computers and Industrial Engineering 42* (2002), 237–242.

[84] Li, J.-P., Balazs, M. E., Parks, G. T., and Clarkson, P. J. A species conserving genetic algorithm for multimodal function optimization. *Evolutionary Computation 10*, 3 (2002), 207–234.

[85] Li, X. Efficient differential evolution using speciation for multimodal function optimization. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO 2005)* (Washington DC, USA, 2005), pp. 873 – 880.

[86] Liang, J., Suganthan, P., and Deb, K. Novel composition test functions for numerical global optimization. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (2005), pp. 68–75.

[87] Lin, C., Punch, W., and Goodman, E. Coarse-grain parallel genetic algorithms: Gategorization and new approach. In *Proceedings of the sixth IEEE Symposium on Parallel and Distributed Processing* (Dallas, USA, 1994), pp. 28–37.

[88] Lloyd, S. Least squares quantization in pcm. *IEEE Transactions on Information Theory 28*, 2 (1982), 129–137.

[89] Lobo, F. G., and Lima, C. On the utility of the multimodal problem generator for assessing the performance of evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006)* (2006), ACM Press.

[90] Locatelli, M. *Handbook of Global Optimization*, vol. 2. Springer, 2002, ch. Simulated Annealing Algorithms for Continuous Global Optimization, pp. 179–229. ISBN 1-40200-632-2.

[91] Lozano, M., Herrera, F., Krasnogor, N., and Molina, D. Real-coded memetic algorithms with crossover hill-climbing. *Evolutionary computation 12*, 3 (2004), 273–302.

[92] Macnish, C. Towards unbiased benchmarking of evolutionary and hybrid algorithms for real-valued optimisation. *Connection Science 19*, 4 (2007), 361–385.

[93] Mahfoud, S. Genetic drift in sharing methods. In *Profeedings of the First IEEE Conference on Evolutionary Computation* (27-29 June 1994), pp. 67–72.

[94] Mahfoud, S. *Niching methods for genetic algorithms.* PhD thesis, university of Illinois, Urbana, Illinois, USA, 1995.

[95] Mahfoud, S. W. Crowding and preselection revisited. In *Parallel Problem Solving from Nature 2* (Amsterdam, 1992), R. Männer and B. Manderick, Eds., Elsevier Science Publishers, B. V., pp. 27–36.

[96] Mahfoud, S. W. A comparison of parallel and sequential niching methods. In *Proceedings of 6th International Conference on Genetic Algorithms* (Pittsburg, USA, July 15-19 1995), pp. 136–143. ISBN 1-55860-370-0.

[97] Martin, W., Lienig, J., and Cohoon, J. *Handbook of Evolutionary computation.* Oxford University Press, 1997, ch. 6.3: Island (Migration) Models: Evolutionary Algorithms based on Punctuated Equilibria.

[98] Menczer, F., and Belew, K. Local selection. In *Proceedings of 7th annual conference on evolutionary programming* (1998), pp. 703–712. ISBN 978-3-540-64891-8.

[99] Mengsheol, O., and Goldberg, D. Probabilistic crowding: deterministic crowding with probabilistic replacement. In *Proceedings of genetic and Evolutionary computation conference (GECCO 1999)* (Orlando, Florida,USA, 13-17 July 1999), pp. 409–416.

[100] Mezura-Montes, E., Vel'aquez-Reyes, J., and Coello Coello, C. A comparative study of differential evolution variants for global optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006)* (Washington, USA, 8-12 July 2006), ACM Press, pp. 485–492.

[101] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, 1996.

[102] Michalewicz, Z., Deb, K., Schmidt, M., and Stidsen, T. Test-case generator for nonlinear continuous parameter optimization techniques. *IEEE Transactions on Evolutionary Computation 4* (2000), 197–215.

[103] Michalewiz, Z., Deb, K., Schmidt, K., and Stidsen, T. Towards understanding constraint-handling methods in evolutionary algorithms. In *Proceedings of the Congress on Evolutionary Computation (CEC 1999)* (Washington DC, 6-9 July 2009), pp. 581–588.

[104] Miller, B., and Shaw, M. Genetic algorithms with dynamic niche sharing for multimodal function optimization. In *Proceedings of 1996 IEEE International congress on Evolutionary Computation (IGEC'96)* (New York, USA, 1996), pp. 786–791.

[105] Moré, J., and Wu, Z. Global continuation for distance geometry problems. *Journal of optimization 7* (1997), 814–836.

[106] Morrison, R. *Designing Evolutionary Algorithms for Dynamic Environments.* Springer-Verlag, Berlin, Germany, 2004.

[107] Morrison, R., and Jong, K. D. A test problem generator for nonstationary evironments. In *Proceedings of the Congress of Evolutionary Computation* (Piscataway, NJ, 1999), IEEE Press, pp. 1843 – 1850.

[108] Murray, W., and Ng, K. *Handbook of Global Optimization*, vol. 2. Springer, 2002, ch. Algorithms for Global optimization and Discrete Problems Based on Methods for Local Optimization,, pp. 87–113. ISBN 1-40200-632-2.

[109] Nelder, J., and Mead, R. A simplex method for function minimization. *The Computer Journal 7*, 5 (1965).

[110] Noman, N., and Iba, H. Enchancing differential evolution performance with local search for high dimensional function optimization. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2005)* (Washington DC, USA, 2005), pp. 967–974.

[111] Noman, N., and Iba, H. Accelerating differential evolution using adaptive local search. *IEEE Transactions on Evolutionary Computation 12*, 1 (2008), 107–125.

[112] Ono, S., Hirotani, Y., and Nakayama, S. Multiple solution search based on hybridization of real-coded evolutionary algorithm and quasi-newton method. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2007)* (Singapore, 25-28 September 2007), pp. 1133–1140.

[113] Parrott, D., and Li, X. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation 10*, 4 (August 2006), 440–458.

[114] Parsapoulos, K., and vrathis, M. On the computation of all minimizers through particle swarm optimization. *IEEE Transactions on Evolutionary Computation 8*, 3 (2004).

[115] Peng, J., Thompson, S., and Li, K. A gradient-guided niching method in genetic algorithm for solving continuous optimization problems. In *proceedings of the 4th Congress on Intelligent Control and Automation* (Shanghai, China, June 10-14 2002), pp. 3333–3338.

[116] Pétrowski, A. A clearing procedure as a niching method for genetic algorithms. In *Proceedings of the 3rd IEEE International Conference on Evolutionary Computation* (Nagoya, Japan, 1996), pp. 798–803.

[117] Pétrowski, A. An efficient hierarchical clustering technique for speciation. Tech. rep., Institute National des Telecommunications, Evry, France, 1997.

[118] Pétrowski, A., and Giros, G. Classification tree for speciation. In *Proceedings of the Congress on Evolutionary Computation (CEC 1999)* (1999), pp. 204–211.

[119] Potter, M., and De Jong, K. Cooperative coevolution: An architecture for evolving coadapted subpopulations. *Evolutionary computation 8*, 1 (2000), 1–29.

[120] Pradeep, K., and Ranjan, G. An automated hybrid genetic-conjugate gradient algorithm for multimodal optimization problems. *Applied Mathematics and Computation 167* (August 2005), 1454–1474.

[121] Press, W., Flannery, B., Teukolsky, S., and Vetterling, W. *Numerical Recipes in C*, second ed. Cambridge University Press, 1992. ISBN 0-521-43108-5.

[122] Price, K. An introduction to differential evolution. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGrawHill, pp. 79–108. ISBN 007-709506-5.

[123] Price, K. *Advances in Differential Evolution*, vol. 143. Springer, 2008, ch. Eliminating Drift Bias from the Differential Evolution Algorithm, pp. 33–88. ISBN 978-3-540-68827-3.

[124] Price, K., and Rönkkönen, J. Comparing the uni-modal scaling performance of global and local selection in mutation-only differential evolution algorithm. In *Proceedings of 2006 IEEE World Congress on Computational Intelligence* (Vancouver, Canada, 16-21 July 2006), pp. 7387–7394. ISBN 0-7803-9489-5.

[125] Price, K., Storn, R., and Lampinen, J. *Differential Evolution: A Practical Approach to Global Optimization.* Springer, 2005. ISBN 3-540-20950-6.

[126] Rigling, B., and Moore, F. Exploitation of sub-populations in evolutionary strategies for improved numerical optimization. In *proceedings of 11th Midewest Artificial Intelligence and Cognitive Science conference (MAICS 1999)* (1999), AAAI press, pp. 80–88.

[127] Robert, C., and Casella, G. *Monte Carlo Statistical Methods*, 2 ed. Springer, 2004.

[128] Roberts, G., and Rosenthal, J. Optimal scaling for various metropolis-gastings algorithms. *Statistical Science 16* (2001), 351–367.

[129] Rosenbrock, H. An automatic method for findong the greatest or least value of a function. *The Computer Journal 3*, 3 (1960), 175–184.

[130] Rudolph, G. Convergence of evolutionary algorithms in general search spaces. In *Proceedings of the third IEEE Conference of Evolutionary Computation* (1996), IEEE Press, pp. 50–54.

[131] Rumpler, J., and Moore, F. Automatic selection of sub-populations and minimal spanning distances for improved numerical optimization. In *Proceedings of the Congress on Evolutionary Computation (CEC 2001)* (2001), vol. 1, pp. 38–43. ISBN: 0-7803-6657-3.

[132] Rönkkönen, J. Normaalijakaumaan perustuva mutaatio-operaatio differentiaalievoluutioalgoritmiin. Master's thesis, Lappeenranta University of Technology, 2003. In Finnish.

[133] Rönkkönen, J., Kukkonen, S., and Lampinen, J. A comparison of differential evolution and generalized generation gap algorithms. *Journal of Advanced Computational Intelligence and Intelligent Informatics 9*, 5 (September 2005). ISSN 1343-0130.

[134] Rönkkönen, J., and Lampinen, J. On using normally distributed mutation step lenght for the differential evolution algorithm. In *Proceedings of 9th MENDEL International Conference on Soft Computing* (Brno, Czech Republic, 4-6 June 2003), pp. 11–18. ISBN 80-214-2411-7.

[135] Rönkkönen, J., and Lampinen, J. An extended mutation concept for the local selection based differential evolution algorithm. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2007)* (London, England, 16-21 July 2007), pp. 689–696. ISBN 978-1-59593-697-4.

[136] Rönkkönen, J., and Lampinen, J. On determining multiple global optima by differential evolution. In *Evolutionary and Deterministic Methods for Design, optimization and Control, Proceedings of Eurogen 2007* (Jyväskylä, Finland, 11-13 June 2007), pp. 146–151. ISBN 978-84-96736-45-0.

[137] Rönkkönen, J., Li, X., Kyrki, V., and Lampinen, J. A generator for multimodal test functions with multiple global optima. In *Proceedings of the 7th International Conference on Simulated Evolution And Learning (SEAL08)* (Melbourne, Australia, 7-10 December 2008), pp. 239–248. ISBN 978-3-540-89693-7.

[138] Rönkkönen, J., Li, X., Kyrki, V., and Lampinen, J. A framework for generating tunable test functions for multimodal optimization. *Soft Computing* (2009). Accepted for publication.

[139] Salomon, R. Reevaluating genetic algorithms performance under coordinate rotation of benchmark functions. *Biosystems 39* (1996), 263–278.

[140] Sareni, B., and krähenbuhl, L. Finess sharing and niching methods revisited. *IEEE transactions in evolutiona computation 2*, 3 (September 1998), 97–106.

[141] Schoen, F. *Handbook of Global Optimization*, vol. 2. Springer, 2002, ch. Two-Phase Methods for Global Optimization, pp. 151–177. ISBN 1-40200-632-2.

[142] Schwefel, H. *Evolution and Optimum Seeking.* Wiley-Interscience, 1995. ISBN 978-0471571483.

[143] Schwefel, H., and Kursawe, F. *Numerical optimization of computer models.* John Wiley & Sons, 1981. ISBN 0471099880.

[144] Shang, Y. *Global Search Methods for Solving Nonlinear optimization problems.* PhD thesis, University of Illinois, Urbana, Illinois, USA, 1997.

[145] Shang, Y., and Qiu, Y. A note on the extended rosenbrock function. *Evolutionary Computation 14*, 1 (2006), 119–126.

[146] Shewchuk, J. An introduction to the conjugate gradient method without the agonizing pain. Tech. Rep. CMUCS-TR-94-125, Carnegie Mellon University, 1994.

[147] Shir, O. *Niching in derandomized evolution strategies and its applications in quantum control.* PhD thesis, Leiden University, 2008.

[148] Shir, O., and Bäck, T. Dynamic niching in evolution strategies with covariance matrix adaptation. In *Proceedings of the Congress on Evolutionary Computation (CEC 2005)* (2005), IEEE press, pp. 2584–2591.

[149] Shir, O., and Bäck, T. Niching in evolution strategies. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO 2005)* (Washington DC, USA, 2005), pp. 915–916.

[150] Shir, O., and Bäck, T. Niche radius adaptation in the cma-es niching algorithm. In *Parallel Problem Solving from Nature 9* (2006), Springer, pp. 142–151. ISBN 978-3-540-38990-3.

[151] Singh, G., and Deb, K. Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Seattle, WA, 2006), ACM Press, pp. 1305–1312.

[152] Smith, R., Forrest, S., and Perelson, A. Searching for diverse cooperative populations with genetic algorithms. *Evolutionary Computation 1*, 2 (1992), 127–149.

[153] Spears, W. Simple subpopulation schemes. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, World Scientific, pp. 296–307.

[154] Stoean, C., Preus, M., Gorunescu, R., and Dumitrescu, D. Elitist generational genetic chromodynamics - a new radii based evolutionary algorithm for multimodal optimization. In *Proceedings of the Congress on Evolutionary Computation (CEC 2005)* (2005), IEEE press, pp. 1839–1846.

[155] Storn, R. On the usage of differential evolution for function optimization. In *Proceedings of the Biennial Conference of the north American Fuzzy Information processing Society* (Berkeley, USA, 19-22 June 1996), IEEE Press, pp. 519–523.

[156] Storn, R. *Advances in Differential Evolution*, vol. 143. Springer, 2008, ch. Differential Evolution Research - Trends and open Questions, pp. 1–31. ISBN 978-3-540-68827-3.

[157] Storn, R., and Price, K. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Tech. Rep. TR-95-012, International Computer Science Institute (ICSI), 1995.

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

[159] Streichert, F., Stein, G., Ulmer, H., and Zell, A. A clustering based niching ea for multimodal search spaces. In *Proceedings of the International Conference Evolution Artificielle* (2003), vol. 2936, Springer, pp. 293–304.

[160] Tawarmalani, M., and Sachinidis, N. *Handbook of Global Optimization*, vol. 2. Springer, 2002, ch. Exact Algorithms for Global optimization of mixed-integer nonlinear programs, pp. 65–85. ISBN 1-40200-632-2.

[161] Ter Braak, C. Genetic algorithms and markov chain monte carlo: Differential evolution markov chain makes bayesian computing easy. Tech. rep., Biometris (part of Wageningen University and Research Centre), 2004.

[162] Ter Braak, C. A markov chain monte carlo version of the genetic algorithm differential evolution: Easy bayesian computing for real parameter spaces. *Statistics and Computing 16*, 3 (2006), 239–249.

[163] Thomsen, R. Multimodal optimization using crowding-based differential evolution. In *Proceedings of the 2004 Congress on Evolutionary Computation* (Portland, 20-23 June 2004), vol. 2, pp. 1382–1389.

[164] Tirronen, V. *Global Optimization Using Memetic Differential Evolution with Applications to Low Level Machine Vision.* PhD thesis, University of Jyväskylä, 2008.

[165] Tirronen, V., Neri, F., Kärkkäinen, T., Majava, K., and Rossi, T. A memetic differential evolution in filter design for defect detection in paper production. *Applications of Evolutionary Computing, Lectures Notes in Computer Science 4448* (2007), 320–329.

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

[167] Törn, A. Cluster analysis using seed points and density determined hyperspheres as an aid to global optimization. *IEEE Transactions on Systems, Man and Cybernetics 7*, 8 (1977), 610–616.

[168] Törn, A., and Zilinskas, A. *Global Optimization, (Lecture Notes in Computer Science).* Springer, 1989. ISBN 9783540508717.

[169] Ursem, R. Multinational evolutionary algorithms. In *Proceedings of Congress of Evolutionary Computation (CEC99)* (1999), vol. 3, IEEE Press.

[170] Ursem, R. Multinational gas: Multimodal optimization techniques in dynamic environments. In *Proceedings of the Second Genetic and Evolutionary Computation Conference (GECCO 2000)* (2000), pp. 19–26.

[171] van den Bergh, F., and Engelbrecht, A. A new locally convergent particle swarm optimizer. In *Proceedings of the 2002 IEEE International Conference on Systems, Man and Cybernetics* (2002). ISBN: 0-7803-7437-1.

[172] Vitela, J., and Castanos, O. A real-coded niching memetic algorithm for continuous multimodal function optimization. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2008)* (Washington, USA, 22-24 July 2008), pp. 2170–2177. ISBN: 978-1-4244-1822-0.

[173] Wei, L., and Mei, Z. A niche hybrid genetic algorithm for global optimization of continuous multimodal functions. *Applied Mathematics and Computation 160* (2005), 649–661.

[174] Weise, T., Niemczyk, S., Skubch, H., Reichle, R., and Geihs, K. A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation* (Atlanta, GA, USA, July 12–16 2008), pp. 795–802.

[175] Whitley, D., lunacek, M., and Sokolov, A. Comparing the niches of cma-es, chc and pattern search using diverse benchmarks. In *Parallel Problem Solving from Nature (PPSN IX)* (2006), Springer, pp. 988–997. ISBN 978-3-540-38990-3.

[176] Whitley, D., Mathias, K., Rana, S., and Dzubera, J. Evaluating evolutionary algorithms. *Artificial Intelligence 85* (1996), 245–276.

[177] Whitley, L., Gordon, S., and Mathias, K. Lamarckian evolution, the baldwin effect and function optimization. In *Parallel Program Solving from Nature 3* (1994), pp. 6–15.

[178] Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics 1* (1945), 80–83.

[179] Wolpert, D., and Macready, W. No free lunch theorems for search. Tech. Rep. SFI-TR-95-02-010, The Santa Fe Institute, 1995.

[180] Wolpert, D., and Macready, W. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1*, 1 (1997), 67–82.

[181] Yin, X., and Germay, N. A fast genetic algorithm with sharing scheme using clustering analysis methods in multimodal function optimization. In *Proceedings of International Conference on Artificial Neural Networks and Genetic Algorithms* (1993), pp. 450–457.

[182] Zaharie, D. Critical values for the control parameters of differential evolution algorithms. In *Proceedings of 8th MENDEL International Conference on Soft Computing* (Brno, Czech Republic, 5-7 June 2002), pp. 62–67. ISBN 80-214-2135-5.

[183] Zaharie, D. Extensions of differential evolution algorithms for multimodal optimization. In *Proceedings of 6th International Symposium of Symbolic and numeric Algorithms for Scientific Computing (SYNASC'04)* (Timisoara, Romania, 26-30 September 2004), pp. 523–534.

[184] Zaharie, D. A multipopulation differential evolution algorithm for multimodal optimization. In *Proceedings of 10th MENDEL International Conference on Soft Computing* (Brno, Czech Republic, 16-18, June 2004), pp. 17–22.

[185] Zelinka, I. Soma - self-organizing migrating algorithm. In *New Optimization Techniques in Engineering*, G. Onwubolu and B. Babu, Eds., vol. 141. Springer, 2004. ISBN 3-540-20167.

[186] Zhang, G., and Lu, H. Hybrid real-coded genetic algorithm with quasi-simplex technique. *International Journal of Computer Science and Network Security 6*, 10 (October 2006), 246–255.

[187] Zhang, J., and Li, K. A sequential niching technique for particle swarm optimization. In *Proceedings of 2005 international conference on intelligent computing, ICIC 2005* (Hefei, China, 23-26 August 2005), pp. 390–399. ISBN 3-540-28226-2.

[188] Ökdem, S. A simple and global optimization algorithm for engineering problems: Differential evolution algorithm. *Turkish Journal of Electrical Engineering & Computer Sciences 12* (2004), 53–60.