University of Southampton

Computational Engineering and Design Group

School of Engineering Sciences

Tuning & Simplifying Heuristical Optimization

by

Magnus Erik Hvass Pedersen

Thesis for the degree of Doctor of Philosophy

January 2010

University of Southampton

Abstract

Computational Engineering and Design Group
School of Engineering Sciences
Doctor of Philosophy

Tuning & Simplifying Heuristical Optimization

by Magnus Erik Hvass Pedersen

This thesis is about the tuning and simplification of black-box (direct-search, derivative-free) optimization methods, which by definition do not use gradient information to guide their search for an optimum but merely need a fitness (cost, error, objective) measure for each candidate solution to the optimization problem. Such optimization methods often have parameters that influence their behaviour and efficacy. A Meta-Optimization technique is presented here for tuning the behavioural parameters of an optimization method by employing an additional layer of optimization. This is used in a number of experiments on two popular optimization methods, Differential Evolution and Particle Swarm Optimization, and unveils the true performance capabilities of an optimizer in different usage scenarios. It is found that state-of-the-art optimizer variants with their supposedly adaptive behavioural parameters do not have a general and consistent performance advantage but are outperformed in several cases by simplified optimizers, if only the behavioural parameters are tuned properly.

Contents

Declaration of Authorship

I, Magnus Erik Hvass Pedersen declare that the thesis entitled Tuning & Simplifying Heuristical Optimization and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- some of the work derived from this thesis has been published, as listed in section 1.9 on page 11.

Signed & Date:

Acknowledgement

First and foremost my project supervisor Andy Chipperfield is thankfully acknowledged for being instrumental in transforming the early drafts of the thesis and papers into a suitable form. Professor Keane, head of the research group, for encouraging me to apply for admission to his group after forwarding to him my first draft of the thesis. Professor Nick Sahinidis and his PhD student at the time, Luis Miguel Rios, for various technical input and Nick for also inviting me to give a conference talk. Dr. Rainer Storn for extensive suggestions and discussions. And last but not least, Professor Emeritus Rein Luus for thorough comments on both the thesis and papers.

Magnus Erik Hvass Pedersen, Denmark, June 2009

Life is really simple, but we insist on making it complicated.

– Confucius

Nomenclature

Abbreviations

DE	Differential Evolution (page 18)
ES	Evolution Strategies (page 6)
GA	Genetic Algorithm (page 5)
GP	Genetic Programming (page 6)
HC	Hill-Climber (page 17)
LUS	Local Unimodal Sampling (page 14)
NFL	No Free Lunch Theorems (page 2)
PS	Pattern Search (page 12)
PSO	Particle Swarm Optimization (page 19)
SA	Simulated Annealing (page 17)

Set & Vector Notation

$\mathbb{N}$	The set of all natural numbers: $\{1, 2, 3, \cdots\}$
$\mathbb{R}$	The set of all real numbers.
$[a, b]$	Set of real numbers between a and b, including endpoints.
(a, b)	Set of real numbers between a and b, excluding endpoints.
$x \in X$	x belongs to the set X
$X_1 \cap X_2$	Intersection, the elements x that are both in X_1 and X_2
$X_1 \times X_2$	Cartesian product, all pairs $[x_1, x_2]$ where $x_1 \in X_1$ and $x_2 \in X_2$
X^n	The set $X \times X \times \cdots \times X$, that is, repeated n times.
$\mathbb{R}^n$	The set of real-valued n-dimensional vectors.
$\vec{v}$	A vector of values, e.g. $\vec{v} = [v_1, v_2, \cdots, v_n]$ for some $n \in \mathbb{N}$
$\|\vec{v}\|$	Length of vector $\vec{v}$, defined as: $\|\vec{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2}$
$C_n(d)$	Hypercube defined as: $C_n(d) = [-d, d]^n$
$B_n(r)$	Hyperball with radius r, defined as: $B_n(r) = \{\vec{v} : \|\vec{v}\| \leq r\}$
$\|x\|$	Absolute value of $x \in \mathbb{R}$.
$\|X\|$	Volume of the set X.
$\sum$	Sum of elements, for example: $\sum_{i=1}^{n} a_i = a_1 + a_2 + \cdots + a_n$

Logical Reasoning

$\forall x \in X : s$	For all elements x in the set X the statement s holds.
$\exists x \in X : s$	There exists at least one element x from X so s holds.
$s_1 \wedge s_2$	Logical *and*. Both statements s_1 and s_2 hold.
$s_1 \vee s_2$	Logical *or*. At least one of s_1 or s_2 holds.
$s_1 \Leftrightarrow s_2$	Bi-implication. Statement s_1 holds if and only if s_2 holds.

Stochastic Notation

$Pr[e]$	Probability of some event e occurring.
$r \sim U(a, b)$	Random variate r, uniformly distributed in the range (a, b)
$\vec{r} \sim U\left(\vec{a}, \vec{b}\right)$	Random vector with each $r_i \sim U(a_i, b_i)$

Optimization Notation

f	Fitness function to be optimized.
∇f	Gradient of function f (defined on page 4)
h	Auxiliary fitness function, e.g. negated or multi-objective.
$x \leftarrow y$	Assign value of y to variable x.
$\vec{b}_{lo}$	Lower boundary of search-space.
$\vec{b}_{up}$	Upper boundary of search-space.

Differential Evolution

$\vec{x}$	An agent's current position in search-space.
$\vec{y}$	Agent's potentially new position in search-space.
$\vec{a}$, $\vec{b}$, $\vec{c}$	Other agents' positions in search-space.
$\vec{g}$	Best discovered position in search-space.
NP	Number of agents in the population.
CR	Crossover probability.
F	Differential weight.
R	Randomly chosen dimension.
r_i	Stochastic variable, $r_i \sim U(0, 1)$

Particle Swarm Optimization

$\vec{x}$	Particle's current position in search-space.
$\vec{v}$	Particle's velocity.
$\vec{p}$	Particle's best discovered position in search-space.
$\vec{g}$	Swarm's best discovered position in search-space.
S	Number of particles in the swarm.
ω	Inertia weight used in velocity update.
ϕ_p	Weight on $\vec{p}$ in velocity update.
ϕ_g	Weight on $\vec{g}$ in velocity update.
r_p, r_g	Stochastic weights used in velocity update, $r_p, r_p \sim U(0,1)$

Chapter 1

Introduction

1.1 What is Optimization?

Candidate solutions to some problems are not simply deemed correct or incorrect but are instead rated in terms of quality and finding the candidate solution with the highest quality is known as optimization.

Optimization problems arise in many real-world scenarios. Take for example the spreading of manure on a cornfield, where depending on the species of grain, the soil quality, expected amount of rain, sunshine and so on, we wish to find the amount and composition of fertilizer that maximizes the crop, while still being within the bounds imposed by environmental law.

1.1.1 Formal Definition

To formalize the concept of optimization consider X to be the set of candidate solutions to the optimization problem. Typically X is n-dimensional over some domain, for example binary: $X = \{0, 1\}^n$, or real-valued: $X \subseteq \mathbb{R}^n$. The domain X is often referred to as the *search-space*. Let the optimization problem be defined by the function f, which is called the fitness function (or cost function, error function, objective function) and rates how well the candidate solutions in X fare on the given problem:

$$f : \ X \to \mathbb{R} \tag{1.1}$$

Without lack of generalization this thesis considers minimization problems, that is, to minimize the fitness function f and hence obtain the candidate solution that fares best, find $x \in X$ so that:

$$\forall y \in X : \ f(x) \leq f(y)$$

Such a point x is known as a global minimum for the function f. It is usually not possible to pinpoint the global minimum exactly in optimization and candidate solutions with sufficiently good fitness are deemed acceptable for practical reasons.

Maximization problems can be optimized merely by introducing an auxiliary function. Suppose f is a fitness function to be maximized, then the analogous minimization problem can be considered instead, simply by introducing the function: $h(x) = -f(x)$

1.2 Challenges

Several challenges arise in optimization. First is the nature of the problem to be optimized which may have several local optima the optimizer can get stuck in, the problem may be discontinuous, candidate solutions may yield different fitness values when evaluated at different times, and there may be constraints as to what candidate solutions are feasible as actual solutions to the real-world problem.

1.2.1 Curse of Dimensionality

Furthermore, the large number of candidate solutions to an optimization problem makes it intractable to consider all candidate solutions in turn, which is the only way to be completely sure that the global optimum has been found. This difficulty grows much worse with increasing dimensionality, which is frequently called the *curse of dimensionality*, a name that is attributed to Bellman, see for example [1, preface p. ix]. This phenomenon can be understood by first considering an n-dimensional binary search-space. Here, adding another dimension to the problem means a doubling of the number of candidate solutions. So the number of candidate solutions grows exponentially with increasing dimensionality. The same principle holds for continuous or real-valued search-spaces, only it is now the volume of the search-space that grows exponentially with increasing dimensionality. In either case it is therefore of great interest to find optimization methods which not only perform well in few dimensions, but do not require an exponential number of fitness evaluations as the dimensionality grows. Preferably such optimization methods have a linear relationship between the dimensionality of the problem and the number of candidate solutions they must evaluate in order to achieve satisfactory results, that is, optimization methods should ideally have linear time-complexity $O(n)$ in the dimensionality n of the problem to be optimized.

1.2.2 No Free Lunch

Another challenge in optimization arises from how much or how little is known about the problem at hand. For example, if the optimization problem is given by a simple formula then it may be possible to derive the inverse of that formula and thus find its optimum. Other families of problems have had specialized methods developed to optimize them efficiently. But when nothing is known about the optimization problem at hand, then the *No Free Lunch* (NFL) set of theorems by Wolpert and Macready [2] state that any one optimization method

will be as likely as any other to find a satisfactory solution. Or as expressed in the following quote from the original paper [2, Page 69]:

> ... if an algorithm performs well on a certain class of problems, then it necessarily pays for that with degraded performance on the set of all remaining problems.

This is especially important in deciding what performance goals one should have when designing new optimization methods, and whether one should attempt to devise the ultimate optimization method which will adapt to all problems and perform well. According to the NFL theorems such an optimization method does not exist and the focus of this thesis will therefore be on the opposite: Simple optimization methods that perform well for a range of problems of interest.

1.3 Classic Optimization

In numerical optimization an initial guess is made at what the optimum might be and this candidate solution is then continually refined until some criterion is met, such as a fitness value that is deemed *good enough* has been reached, or a certain number of iterations have been performed.

1.3.1 Newton-Raphson

Perhaps the most classic form of numerical optimization is by way of the *Newton-Raphson* iteration (see e.g. [3, p. 81]) which produces a sequence of values $x \in \mathbb{R}$ in an attempt to find a root of some single-dimensional function $h : \mathbb{R} \to \mathbb{R}$:

$$x \leftarrow x - \frac{h(x)}{h'(x)} \tag{1.2}$$

with the initial x being some reasonable guess. For some functions h the initial value x may be chosen sufficiently close to the root, thus guaranteeing the root is found. Then consider a fitness function $f : \mathbb{R} \to \mathbb{R}$ with a single variable that must be optimized. This can then be done with the Newton-Raphson root-finding method by setting $h(x) = f'(x)$ and finding the roots of f':

$$x \leftarrow x - \frac{f'(x)}{f''(x)} \tag{1.3}$$

A root of f' is also called a critical point of f and is either a local optima of f or a saddle point. Whichever critical point of f that is found (assuming it has more than one) depends on the starting position of x. This method requires that f' and f'' exist, and moreover that $f''(x) \neq 0$ for all x. The derivative h' in Eq.(1.2) (or f'' in Eq.(1.3)) can also be estimated from close points by using finite difference approximation and the Newton-Raphson root-finding method is then called the Secant method. Though using the Secant method for optimization still requires the first order derivative.

1.3.2 Quasi-Newton Methods

Generalizing the Newton-Raphson method to multi-dimensional search-spaces, that is, to fitness functions of the form $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and assumed to be twice differentiable, would change Eq.(1.3) to:

$$\vec{x} \leftarrow \vec{x} - [Hf(\vec{x})]^{-1} \nabla f(\vec{x})$$

where ∇f is called the gradient and is the vector of f's first order partial derivatives, with the derivative in regards to the variable for the i'th dimension x_i being denoted $\partial f/\partial x_i$. Differentiating with regards to the variable for each dimension the n-dimensional gradient is found thus:

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n} \right]^T \tag{1.4}$$

The matrix $[Hf(\vec{x})]^{-1}$ is the inverse of the Hessian matrix for the fitness function f, consisting of its second order partial differentials, that is, the ij'th entry of the Hessian matrix equals: $(Hf)_{ij} = \partial^2 f/\partial x_i \partial x_j$.

The gradient ∇f can be difficult to derive and the Hessian Hf is usually even more difficult to derive. It is therefore customary to use Quasi-Newton optimization methods which seek to approximate the Hessian in a recurrent manner during the optimization iterations. One of the earliest quasi-Newton methods is called the *Davidon, Fletcher,* and *Powell* (DFP) formula and is belatedly documented in [4]. Another similar but perhaps more popular quasi-Newton method was developed independently by the researchers Broyden [5], Fletcher [6], Goldfarb [7], and Shanno [8], and called the BFGS method. These quasi-Newton methods, however, not only require the existence of the gradient in order to optimize a problem but they are also complex to describe and implement.

1.3.3 Gradient Descent Methods

Instead of approximating the Hessian as done in quasi-Newton methods the gradient $\nabla f(\vec{x})$ can be followed in small steps to approach the optimum since the gradient points in the direction of steepest ascent for the fitness function f at the position $\vec{x}$. For minimization problems the reverse direction must be followed, that is, the direction of steepest descent, and the optimization method is hence known as Gradient Descent of which there are numerous variants and a detailed exposition can be found in [9].

1.4 Black-Box Optimization

When the gradient of an optimization problem is unknown, perhaps because it cannot be defined due to a partially discontinuous fitness function, or because the fitness measure changes over time, or perhaps even because the fitness is undefined for certain regions of the search-space, then another kind of optimization method must be used, a kind of method which treats the fitness function

to be optimized as a *black box* that merely produces some measure of fitness given a candidate solution. All that such optimization methods can be guided by is the fitness value at different positions in the search-space at the time of evaluation.

The terminology and taxonomy within this field changes with different researchers and should not be considered too rigid. The following are a few examples of typical but more or less synonymous terms. The terms *Derivative-Free*, *Direct Search*, *Black-Box*, or *Heuristical* optimization methods all refer to methods which do not rely on the gradient of the problem to be optimized in order to guide their search. Another term frequently used is *Meta-Heuristic* which is not used in this thesis however as it collides with the term *Meta-Optimization* that has a very different meaning here. Stochastic methods are also sometimes called *Monte Carlo* methods, named after the European city and casino because of the element of random chance in achieving success with such methods.

When an optimization method employs multiple candidate solutions that are somehow combined to generate new candidate solutions, here often called agents or individuals, the method may be called *multi-agent*, *population-* or *swarm-*based, or *evolutionary*, where the latter typically requires that some kind of selection or survival of most fit individuals take place. Again, this terminology should not be considered too rigid.

1.4.1 Altimetric Analogy

Heuristic function minimization can be likened to the discovery of the deepest point in the deepest valley in a 3-dimensional landscape. Imagine the optimizing agents being placed in one of Mother Nature's landscapes with each agent only being able to do three things: 1) Measure the altitude of its current location, 2) communicate with the other agents, and 3) move to some other location. That is, an agent has no vision and therefore does not know the altitude of its nearby surroundings but only of its current location and possibly also of a finite number of previous locations that it has visited.

Since there are infinitely many different points where an optimizing agent can go to, each agent will have to choose its path carefully as it can only visit a finite number of points in a finite amount of time. The question is then, what should the agents communicate to each other, and how should they move around in the landscape?

1.4.2 Genetic Algorithm

An early multi-agent, black-box optimization method is known as the *Genetic Algorithm* (GA) and was inspired by evolution of biological individuals which makes the individuals adapt to their environment through generations; a theory that was originally proposed by Darwin [10]. The GA for doing numerical optimization is attributed to several sources, a popular one being Holland [11]. Another, perhaps more practical text on GA is due to Goldberg [12].

The GA works by selecting individuals from its population, that is, selecting candidate solutions to the optimization problem from its current pool according to their relative fitness. Some of these individuals simply survive while others reproduce to form new individuals. Furthermore, random alterations to the individuals account for the mutation occurring in Mother Nature. That some individuals do not make it to the next generation can be likened to the everlasting fight for natural resources and nature's way of ridding itself from weak or ill-suited individuals, and eventually entire species.

The GA was originally developed for binary search-spaces (see for example Goldberg [12]), but optimization over real-valued search-spaces is also possible. This has been studied primarily for the *Evolution Strategies* (ES) family of optimization methods, which uses evolutionary concepts similar to those of the GA. The original work on ES is in German but an English-languaged survey by Bäck et al. can be found in [13]. Some experiments with real-valued operators for the GA have also been made in the literature and a survey by Herrera et al. can be found in [14].

The abstract GA operators of selection, crossover and mutation can also be used to optimize in more complex search-spaces for which gradients can obviously not be derived, such as the set of all possible computer programs. A popular optimization method for such complex and generic search-spaces is known as *Genetic Programming* (GP) and is due to Koza [15].

1.4.3 Differential Evolution & Particle Swarm Optimization

This thesis will study variants of two popular optimization methods, the first one is known as *Differential Evolution* (DE) and is originally due to Storn and Price [16] [17]. DE is conceptually similar to GA in its use of evolutionary operators to guide the search for an optimum, only DE was specifically developed for real-valued search-spaces from its inception. The other method to be studied in this thesis is called *Particle Swarm Optimization* (PSO) and is originally due to Kennedy, Eberhart and Shi [18] [19], and was originally intended as a model for social behaviour in a flock of birds, but the algorithm was simplified and it was realized that it was actually performing optimization. The DE and PSO methods will be detailed in chapter 2 and studied more rigorously in chapters 4 and 5.

1.4.4 Convergence

It would be preferable to have mathematical proof that a black-box optimizer converges to the global optimum for e.g. certain classes of continuous functions. In the literature such proofs are sometimes made in the form of limit-proofs, in which the optimizer is proven to eventually find an arbitrarily small region surrounding the optimum, provided the optimizer is given enough iterations, see for example [20] [21]. But the exact same thing can be proven for completely random sampling of the search-space, because the probability p of sampling any

non-empty region surrounding the optimum will be non-zero, that is, $p > 0$, and hence the expected number of iterations to reach this region surrounding the optimum will be $1/p$, so the probability for eventually sampling this region will be 1. Convergence proofs that rely on infinity-limits are therefore of no practical use. As mentioned previously, the only way to be certain that the global optimum has indeed been found is to test every single position in the search-space. For real-valued search-spaces there are infinitely many candidate solutions in the search-space, but even if a discrete grid is being traversed the Curse of Dimensionality quickly makes it intractable to perform an exhaustive search of even modestly sized and dimensioned optimization problems. In developing heuristical optimizers they should therefore be tested empirically, so as to instill an adequate degree of confidence in their capabilities in optimizing new problems.

1.5 Tuning Heuristical Optimization

Optimization methods often have parameters that influence their behaviour and efficacy in optimizing different problems. These behavioural parameters can be modified by a practitioner who seeks to improve performance on particular problems of interest to him. Take for instance GA which has parameters for the crossover and mutation rates as well as the population size.

Behavioural parameters have traditionally been chosen according to guidelines compiled by researchers and practitioners through years of experience, for advice on setting the behavioural parameters of PSO see e.g. Shi and Eberhart [22] [23] or the more comprehensive survey by Carlisle and Dozier [24], and for advice on setting the behavioural parameters of DE see e.g. Storn et al. [17] [25], and Liu and Lampinen [26]. However, these guidelines are based on human experience and coarse experiments with combinations of parameter settings, which are often biased by what the researchers believe makes the heuristical optimizers work well, often ignoring certain parameter combinations which are believed *a priori* to yield poor performance. Several examples of this will be given in later chapters of this thesis where behavioural parameters that perform very well are being discovered, but they are in violation of common advice given in the literature.

Behavioural parameters may also be selected according to mathematical analysis, see e.g. van den Bergh [27], Trelea [28], and Clerc and Kennedy [29] for analyses of PSO parameters, or Zaharie [30] for analysis of DE parameters. But these analyses make many assumptions that limit their validity and their advice for the selection of behavioural parameters will also be disputed in later chapters.

1.5.1 Meta-Optimization

Finding a good choice of behavioural parameters can instead be considered a form of overlaying optimization problem that can be solved in an *offline* manner,

by having an overlaying optimizer find behavioural parameters for the base-level optimizer so as to make it perform well. The behavioural parameters discovered thus can be used by other practitioners without alterations to the optimization algorithm. Finding behavioural parameters by use of an overlaying optimizer is known here as Meta-Optimization, but is also known in the literature as Meta-Evolution, Super-Optimization, Automated Parameter Calibration, etc. An early attempt at automatically tuning the behavioural parameters of a GA by employing another overlaying optimizer is due to Grefenstette [31]. Experiments with tuning the behavioural parameters of PSO have been conducted by Meissner et al. [32]. A recent comparison of some of the available approaches to meta-optimization is made by Smit and Eiben [33].

However, a challenge that arises in meta-optimization is the immense amount of computational resources needed for making repeated optimizations with new behavioural parameters. This means that past experiments have been of a limited nature concerning the number of optimization problems for which they can simultaneously tune the behavioural parameters, and limited concerning the number of iterations allowed for the meta-optimizer. This has had implications for the quality of results obtained and for the direct usability of the techniques. Both these issues will be addressed in chapter 3.

1.5.2 Adaptation of Behavioural Parameters

Another research trend has been to devise optimizer variants that can adapt their behavioural parameters during optimization in an *online* manner. Despite the NFL theorems stating that there does not exist any ultimate optimizer which can adapt to whatever problem it is posed with, these optimizer variants appear to be most popular in the research literature.

Adaptation of behavioural parameters has been an integral part of the ES family of optimization methods [13] where some variants employ a technique dubbed *Self-Adaptation*, consisting of adding some or all of the optimizers behavioural parameters to the search-space, thus making them subject to optimization along with the problem at hand. This technique is also used for a GA in [34] and [35].

Another way of trying to adapt the behavioural parameters in an online manner during optimization can be found in the Fuzzy Adaptive DE (FADE) by Liu and Lampinen [36], which uses Fuzzy Logic to adapt the DE behavioural parameters during an optimization run. Fuzzy logic, originally due to Zadeh [37], provides a means for logical reasoning with uncertainties and is used in FADE to alter the behavioural parameters according to optimization progress. Another example of an adaptive DE variant is the Self-adaptive DE (SaDE) due to Qin and Suganthan [38]. Yet another DE variant with adaptive behavioural parameters is known as JDE and is due to Brest et al. [39].

Attempts with what might be called *Meta-Adaptation* also appear in the literature, in which an overlaying optimizer is trying to tune the parameters of another optimizer in an online manner during the optimization of a problem. Different versions of this have been developed, see for example [40, Section 10] for

adaptation of PSO parameters during optimization by employing an overlaying DE optimizer.

At a first glance, these schemes may appear to get rid of behavioural parameters altogether, but they usually just introduce new parameters the user must decide upon, such as the endpoints of the adaptive parameter ranges. There seems to be a belief amongst some researchers that even though there are now more behavioural parameters to select, they are easier to select by a user because they have more lenient influence on optimization performance. This will be disputed in later chapters.

Another belief amongst researchers seems to be that different choices of parameters are needed at different stages of optimization, so as to adjust between exploration and exploitation of the search-space. Otherwise the parameters could just as well be held fixed during optimization. This belief will also be disputed in later chapters, but it should be noted here that it seems to be a paradox. Biasing behavioural parameters during optimization towards the parameters that have been observed to work well seems to be contradictory to the need of having different parameters at different stages of optimization. If anything, the behavioural parameters should be changed so as to be dissimilar to the parameters that have previously worked well during that optimization run.

Furthermore, there is an array of other questions one must decide upon to make an implementation of adaptive behavioural parameters, such as: How are the parameters to be initialized? How many optimization iterations should be performed between modifying the parameters? How is the quality of a change in parameters to be rated? Etc. These questions are echoed in [35] whose experimental results indicate self-adaptation is of little use in a GA. Problematic issues regarding self-adaptation and indeed adaptation of behavioural parameters in general are also noted for the ES family of optimization methods in [41].

1.6 Simplifying Heuristical Optimization

Another problematic aspect with these so-called adaptive optimizer variants is that they all increase the algorithmic complexity in an effort to increase adaptability to new optimization problems. This is perhaps a dangerous path to follow because heuristical optimization methods cannot be proven correct analytically. A good example of this dilemma is the Stochastic Genetic Algorithm (StGA) by Tu and Lu [42] which extends and tries to improve upon the basic GA. While StGA did show performance improvement over other optimizers on a suite of benchmark problems, it eventually turned out the StGA implementation had an error that made it strongly biased towards finding the global optima of the benchmark problems considered [43]. Again, such issues arise from the fact that a heuristical optimizer cannot be proven correct by analytical means, and the more complex an optimization method becomes, the harder it gets to describe the method clearly and hence make a correct implementation.

Increasing the algorithmic complexity of a heuristical optimizer is also contrary to the original idea of Self-Organization as it occurs in nature, in which simple individuals cooperate and complex collective behaviour emerges. See for example Hawking's introductions to modern physics [44] [45] and the works of Ball [46], Holland [47, 48], Johnson [49], and Hofstadter [50], just to name a few, on self-organization and emergence in such diverse areas as real ant colonies, brains and human cities.

In this thesis it will be attempted to simplify the DE and PSO optimizers without impairing their performance. This is more in vein of the original idea of self-organization, and also in vein of Occam's Razor; *lex parsimoniae*, or the Law of Parsimony which popularly states that simplicity is usually better.

Simplifying optimizers is not common in the research literature and only a small number of papers seem to have been published on this topic. For instance, Kennedy [51] studied simplifications to the PSO method but unfortunately did not have the tools necessary to make a rigorous comparison with the basic PSO, something which will be done in this thesis because meta-optimization allows for the discovery and comparison of core performance capabilities between optimizer variants. More recent experiments with simplifying PSO are due to Bratton and Blackwell [52] whose research was done concurrently with the work here and has therefore not influenced this thesis. They make use of performance landscapes to study parameter choices for their PSO simplifications and do not employ meta-optimization to discover good performing parameters, which is done in this thesis and allows for more extensive studies as much less computation time is needed.

1.7 Main Contributions

The first contribution of the thesis is found in chapter 3 and is a technique for doing Meta-Optimization so as to find behavioural parameters for an optimizer that makes it perform well. The technique is simple to describe and implement, and is comparatively efficient without sacrificing quality of results.

The other main contribution is found in chapter 4 and comes from using meta-optimization to make a range of experiments with state-of-the-art DE variants that either perturb or adapt their behavioural parameters during optimization, which was previously believed to give a performance advantage over the basic DE. The results show that there does not appear to be a general or consistent advantage to such parameter adaptation. In fact, a simplified DE is introduced which sometimes performs better. Chapter 5 has similar experiments with a simplified PSO variant which turns out to be an overall improvement to the basic PSO from which it was derived.

1.8 Thesis Overview

The thesis is structured as follows:

- Chapter 2 details the DE and PSO methods along with other optimization methods that will be used in the thesis.
- Chapter 3 details the technique for doing Meta-Optimization, that is, the tuning of an optimizer's behavioural parameters by employing another overlaying meta-optimizer in an offline manner.
- Chapter 4 uses meta-optimization to fairly compare DE variants.
- Chapter 5 uses meta-optimization to fairly compare PSO variants.
- Chapter 6 gives an overall conclusion for the thesis along with a number of ideas for future research.

1.9 Publications & Derived Work

Journal publication:

- *Simplifying particle swarm optimization*, Applied Soft Computing, 10:618-628, 2010.

Technical reports:

- *Tuning differential evolution for artificial neural networks*, Hvass Labs., HL0803, 2008.
- *Parameter tuning versus adaptation: proof of principle study on differential evolution.*, Hvass Labs., HL0802, 2008.
- *Local unimodal sampling*, Hvass Labs., HL0801, 2008.

Other derived work that can be mentioned:

- The *SwarmOps* source-code libraries for the C# and ANSI C programming languages which implement the meta-optimization technique as well as the optimizers used in the thesis. The libraries are available for download through the Internet: `http://www.hvass-labs.org/`
- The LUS optimization method from chapter 2 was used by Pellarin [53] in his master's thesis for doing user-rated optimization, in which the fitness function is replaced by a human user who determines which of two candidate solutions is best.

Chapter 2

Optimization Methods

2.1 Introduction

This chapter describes the optimization methods that will be studied in this thesis, as well as the benchmark problems used for empirically evaluating and comparing the performance of the optimizers.

2.2 Pattern Search

In this section a simple optimization method is presented that samples the search-space locally from the current position, and decreases its sampling-range upon failure to improve the fitness. Appendix A gives the motivation for decreasing the sampling-range during optimization, namely that a fixed sampling-range cannot converge to a local optimum. The optimization method presented here was unknowingly reinvented during this research, but turned out to belong to a family of optimizers known under the name *Pattern Search* (PS). The variant presented here is slightly simpler, though.

2.2.1 Related Work

An early variant of PS is from the 1950's and is attributed to the researchers Fermi and Metropolis at the Los Alamos nuclear laboratory. It is described by Davidon [4] as follows:

> ... They varied one theoretical parameter at a time by steps of the same magnitude, and when no such increase or decrease in any one parameter further improved the fit to the experimental data, they halved the step size and repeated the process until the steps were deemed sufficiently small. ...

A similar idea is described by Hooke and Jeeves [54] who also coined the name Pattern Search.

The underlying idea of PS is somewhat related to that of *Golden Section Search* (GSS) due to Kiefer [55], which works for one-dimensional search-spaces by maintaining three separate points, and at each iteration replacing one of these with an intermediate point that is chosen so as to close in on the optimum of a unimodal problem. This concept was generalized for multi-dimensional search-spaces in the *Simplex* optimization method due to Nelder and Mead [56], which for n-dimensional search-spaces requires for $n+1$ positions in the search-space to be maintained. This conceptual idea of closing in around a unimodal optimum is also the basis for the PS variant presented here, but without the need for maintaining several positions in the search-space.

Convergence to global optima of certain well-behaved functions is claimed by Torczon et al. [57] [58] for a family of PS methods. However, the experimental results towards the end of this chapter suggest that these convergence proofs are primarily of theoretical interest, as the simple PS variant given here is not always able to locate the optimum of the benchmark problems.

2.2.2 Sampling

The PS variant given here only samples and changes the value of one dimension at a time, and is therefore able to ascertain whether that change was directly responsible for improvement to the fitness or not, hence allowing it to adapt the sampling range and direction appropriately for just that dimension. The dimension to be sampled is chosen at random in each iteration, which makes for a particularly simple implementation.

The other feature particular to the PS variant presented here, is to start out sampling the full search-space and maintain the sampling direction for each dimension until it no longer yields improvement to the fitness, upon which the sampling direction for that dimension is reverted and the range halved.

By combining these two principles the PS method implicitly tries to follow the gradient of the fitness function in each of its dimensions. The PS method can therefore be considered a black-box version of Gradient Descent.

When dealing with unimodal problems that have only one local optimum, then because the sampling range of PS is initially the full search-space, each iteration of PS will either cause a large improvement to the fitness or it will decrease the sampling range significantly. Either case means big progress. In its exponential halving of the search-space the PS method can be said to resemble a classic bisection algorithm (see for example [59, section 6.2]).

2.2.3 Algorithm

The algorithm for PS is shown in figure 2.1. Note that it is important the update-rule is strictly greedy, in the sense that a less-than comparison is used and not a less-than-or-equal comparison. If the latter was used, then PS could get stuck in the boundaries of the search-space, because PS could continue to sample and accept the same boundary position as an improvement, thus never allowing the sampling-range to decrease and the search direction to reverse.

2.3 Local Unimodal Sampling

In this section a heuristical optimization method called *Local Unimodal Sampling* (LUS)[1] is presented, which can be thought of as an extension of the PS method in that it samples all dimensions simultaneously, while decreasing its sampling-range in much the same manner as PS. Again, appendix A gives the motivation for decreasing the sampling-range during optimization, namely that a fixed sampling-range has no possibility of converging to a local optimum. LUS is so named, because it was designed to optimize unimodal problems, which it is found to do very well indeed, in the experiments on the Sphere benchmark problem towards the end of this chapter.

As with the PS optimizer above the LUS method was reinvented here and similar methods from the literature are surveyed towards the end of the section, when their inner workings can be compared to LUS.

2.3.1 Sampling

For the sampling done by the LUS method, denote by $\vec{y}$ the new potential position chosen from the neighbourhood of the current position $\vec{x}$:

$$\vec{y} = \vec{x} + \vec{a}$$

where the vector $\vec{a}$ is picked randomly and uniformly:

$$\vec{a} \sim U\left(-\vec{d}, \vec{d}\right)$$

where $\vec{d}$ is the current sampling-range, initially chosen as the full range of the search-space and decreased during optimization as described next.

2.3.2 Sampling-Range Decrease

When a sample fails to improve the fitness, the sampling-range is decreased for all dimensions simultaneously. The question is how much this decrease should be? For the PS method presented above, every dimension of an n-dimensional search-space would be halved after n failures to improve the fitness. So it seems reasonable to make LUS have a similar combined effect of halving the sampling-range for every dimension, after n failures to improve its fitness. The sampling-range $\vec{d}$ should therefore be multiplied with q for each failure to improve the fitness:

$$\vec{d} \leftarrow q \cdot \vec{d}$$

with q being defined as:

$$q = \sqrt[n]{1/2}$$

where n is the dimensionality of the problem to be optimized. However, experiments that will be omitted here suggest that it is necessary to make a small

[1] Danish for *louse*.

adjustment to this halving, so the sampling-range decrease-factor is instead given by:

$$q = \sqrt[n/\alpha]{1/2} \;\Leftrightarrow\; q = 2^{-\alpha/n} \tag{2.1}$$

Where $0 < \alpha < 1$ causes slower decrease of the sampling-range, and $\alpha > 1$ causes more rapid decrease. Note that applying this n times yields a sampling-range reduction of $q^n = 1/2^\alpha$ as desired, and for $\alpha = 1$ this would mean a halving of the sampling-range for all dimensions. In the experiments here, a value of $\alpha = 1/3$ is used because it has been found to yield good performance on a wide range of problems. The LUS algorithm is shown in figure 2.2.

2.3.3 Related Work

The following is a survey of optimization methods related to LUS. The important thing to note is that various attempts have been made in the literature at performing local sampling with adaptation of the sampling range. But LUS offers an effective yet simple way of doing this, which is one of the reasons why LUS remains an optimization method of choice for later parts of this thesis.

Surface Sampling

An early form of local sampling of a real-valued optimization problem was suggested in the 1960's by Rastrigin [60]. Rastrigin's basic method may be called *Fixed Step Size Random Search* (FSSRS). The method samples the new potential position from the surface of a hypersphere surrounding the current position, with the diameter of this hypersphere remaining fixed throughout the optimization run. Although Rastrigin claims the FSSRS converges faster than gradient descent towards the optimum of the Sphere function, it should be noted that the FSSRS will not be able to actually converge to the optimum, due to its use of a fixed step size, see appendix A.

The *Optimum Step Size Random Search* (OSSRS) [61] [62] is primarily a theoretical study of how to optimally adjust the step size of the FSSRS, so as to allow for speedy convergence to the optimum. But an actual implementation of the OSSRS needs to approximate this optimal step size by repeated sampling and is therefore expensive to execute.

Instead, the *Adaptive Step Size Random Search* (ASSRS) by Schumer and Steiglitz [62] is a more practical version of the OSSRS, which attempts to heuristically adapt the step size. The algorithm however, is somewhat complicated.

Lawrence and Steiglitz suggest combining the concept of Rastrigin's hyperspherical sampling with a halving of the step size as done in Pattern Search [63]. But since the new method samples all dimensions simultaneously, they slow down the decrease in step size by introducing an inner loop to the algorithm. The inner loop performs n individual samples for an n-dimensional problem, resulting in a halving of the step size only after each of these samples fail to improve the fitness. This bears a perhaps slightly coarse resemblance to the underlying idea of LUS.

It was later realized by Schrack and Choit [64] that the optimal step size could actually be approximated by a simple exponential decrease. This led to the *Optimized Relative Step Size Random Search* (ORSSRS) method. Again, the exponential decrease of the step size resembles how LUS decreases its sampling-range. Although, the formula of the ORSSRS for computing its step size decrease factor is somewhat complicated – in stark contrast to the simple Eq.(2.1) for LUS. The LUS method is therefore preferred over the ORSSRS.

Neighbourhood Sampling

The Rastrigin family of optimization methods that were just surveyed, all sample the surface of a hypersphere surrounding the current position. Another early kind of local sampling is due to Matyas [20] and uses Gaussian distributed sampling of the neighbourhood of the current position. Matyas' method is somewhat similar to the local sampling used in this chapter, with the main difference being that uniform sampling of the neighbourhood is used here.

Matyas claims his basic form of local sampling converges to the optimum of the Sphere function; although it may take a very large number of samples. In fact, Matyas uses a limit-proof which shows convergence to the optimum is certain to occur if an infinite number of samples are made. But this kind of proof also works for purely random sampling of the search-space, as it too will eventually find a position sufficiently close to the optimum (see the brief discussion in the introductory chapter 1.) Matyas' claim is therefore purely theoretical and not practical.

Matyas extends the basic form of local sampling to include adaptation of the sampling range and direction, so the sampling is continued along a successful path until it no longer yields improvement to the fitness, at which time another direction may be sampled. However, the exact algorithm is somewhat intricate, and Matyas also neglects to supply formulae for determining suitable parameters of this adaptation.

Mathematical analyses are also conducted by Baba [65] and Solis and Wets [66] to establish that convergence to a region surrounding the actual optimum is inevitable for different variants of Matyas' local sampling method, under some mild conditions. An estimate on the number of iterations required to approach the optimum is also derived by Dorea [67]. These analyses, however, are criticized through empirical experiments by Sarma [68], who used the optimization methods of Baba and Dorea on two real-world problems, showing the optimum to be approached very slowly, and moreover that the methods were actually unable to locate a solution of adequate fitness, unless the process was started sufficiently close to the optimum to begin with.

While it has not been proven whether the LUS method is truly capable of converging to a local optimum, it has proven effective in practice as will be seen from the experiments in chapters 3, 4, and 5. As the LUS method is also very simple, it is preferred over these other variants.

The Luus-Jaakola Method

The *Luus-Jaakola* (LJ) method from [69] is conceptually very similar to the LUS method that was presented here, in that they both sample from a hypercubical neighbourhood surrounding the current solution, and use a slow exponential decrease of the size of this hypercube. The LJ method, however, was originally suggested to be used with a constant decrease factor of $q = 0.95$, regardless of the dimensionality n of the problem to be optimized. This fixed choice of q corresponds roughly to $n = 5$ dimensions for the LUS method. But recall that the LUS method varies the factor q with increasing n, so as to slow down the sampling-range decrease for higher dimensional problems and thereby take into account the increased difficulty in optimizing such problems.

The issue of using a fixed decrease factor q for the LJ method on higher dimensional problems was studied using probability theory by Nair [70]. It was found that for an optimization problem sufficiently resembling the Sphere function, and for a proper choice of sampling-range decrease factor q, the LJ method does converge to the optimum of that problem. But the decrease factor q must vary with increasing dimensionality n, so as to approach the optimum more slowly for higher dimensional problems. Nair however, does not give explicit formulae for actually calculating the factor q, but merely provides implicit conditions that must be met, and the work is therefore less useful in practice.

Several variants of the LJ method have been developed including [71] [72], and although they all improve on the performance of the original LJ method for different testbeds of optimization problems, they also increase the complexity of the optimization method. Since this thesis is concerned with finding ever simpler and more graceful optimization methods, and the LUS method is in fact more easy to describe and implement than the recent variants of the LJ method, while still being able to perform well on optimization problems of widely ranging topologies and dimensionalities, the LUS method is preferred.

Stochastic Update Rules

In an attempt to make local sampling methods able to escape local optima, some researchers have tried using stochastic update rules when considering whether to move to a newly sampled position. One such update rule is originally due to Metropolis et al. [73] and sometimes goes under the name Stochastic Hill-Climber (HC) [74]. Another stochastic update rule was originally suggested by Kirkpatrick et al. [75] and is known as Simulated Annealing (SA). Stochastic update rules such as these have been used by different researchers for optimization over real-valued search-spaces using local sampling, see e.g. [76] [77].

But as proven mathematically in appendix A, the main problem with local sampling over real-valued search-spaces seems to be in having a fixed sampling-range. In fact, a stochastic update rule does not provide any real chance of escaping a local optimum, because it merely means the optimizer will repeatedly move slightly towards and then slightly away from the local optimum, without ever being able to fully converge to an optimum, nor fully move away from

local optima. To see this, let p be the probability of moving to a position in the search-space having worse fitness. If the sampling-range is such that k consecutive samples will be needed in order to escape a local optima, then the probability of this happening is p^k, which quickly approaches zero as either p becomes small and/or k becomes large. This ignored the fact that not all of k consecutive samples will be away from the local optimum, which means the real number of samples needed is actually much larger and hence the probability of moving away from the local optimum is much smaller. Stochastic update rules should therefore not be expected to improve performance in local sampling methods, but can actually be expected to worsen the optimization performance, as many iterations will be spent on dithering around local optima.

2.4 Differential Evolution

The multi-agent heuristical optimization method known as *Differential Evolution* (DE) is due to Storn and Price [16] [17], and works by creating a new potential agent-position by combining the positions of randomly chosen agents from its population, and updating the agent's current position in case of improvement to its fitness.

2.4.1 Combined Mutation & Crossover

Like GA, the DE method also employs operators that are dubbed *crossover* and *mutation* (albeit with different meanings), and which are typically applied in turn. In the following, however, these operators have been combined for a more concise description.

This study will initially use the DE/rand/1/bin variant because it is believed to be the best performing and hence the most popular of the basic DE variants [25] [78]. Other DE variants will be surveyed and studied in chapter 4.

Let $\vec{y}$ be the new potential position for an agent whose current position is $\vec{x}$, and let $\vec{a}$, $\vec{b}$ and $\vec{c}$ be the positions of three randomly chosen agents that must be distinct from each other as well as the agent $\vec{x}$ currently being processed. The elements of $\vec{y} = [y_1, \cdots, y_n]$ are then computed as follows:

$$y_i = \begin{cases} a_i + F(b_i - c_i) & , (i = R) \text{ or } (r_i < CR) \\ x_i & , \text{else} \end{cases} \tag{2.2}$$

where $F \in [0, 2]$ is a behavioural parameter called the differential weight, and the randomly chosen index $R \in \{1, \cdots, n\}$ ensures at least one element differs from that of the original agent: $y_R \neq x_R$, while the rest of the elements are either chosen from the original position $\vec{x}$ or computed from combining other agents' positions according to the so-called crossover probability $CR \in [0, 1]$, and with $r_i \sim U(0, 1)$ being a uniform random number drawn for each use. An additional behavioural parameter is the number of agents in the population, which for DE is commonly denoted NP. Once the new potential position $\vec{y}$

has been computed, it will replace the agent's original position $\vec{x}$ in case of improvement to the fitness.

The DE algorithm is shown in figure 2.3. Although the informal description above is fairly simple, the actual implementation is somewhat involved.

2.5 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a multi-agent heuristical optimization method due to Kennedy and Eberhart [18]. The PSO method was originally intended for simulating the social behaviour of a bird flock, but the algorithm was simplified and it was realized that the agents – here typically called particles – were actually performing black-box optimization. In PSO the population of particles is typically called a swarm.

In the PSO method the particles are initially placed at random positions in the search-space, moving in randomly defined directions. The direction of a particle is then gradually changed so it will start to move in the direction of the best previous positions of itself and its peers, searching in their vicinity and potentially discovering even better positions. The following details a basic PSO and chapter 5 surveys other PSO variants.

2.5.1 Velocity & Position Update

Let the position of a particle be denoted $\vec{x}$ and let $\vec{v}$ be its velocity. Both are initially chosen randomly and then iteratively updated according to two formulae. The following formula for updating the particle's velocity is by Shi and Eberhart [19]:

$$\vec{v} \leftarrow \omega\vec{v} + \phi_p r_p(\vec{p} - \vec{x}) + \phi_g r_g(\vec{g} - \vec{x}) \tag{2.3}$$

where the behavioural parameter $\omega \in \mathbb{R}$ is called the inertia weight. The particle's best discovered position is $\vec{p}$, and $\vec{g}$ is the swarm's best discovered position through which the particles communicate implicitly with each other. That is, $\vec{g}$ is the best of all the $\vec{p}$'s. These are weighted by the stochastic variables $r_p, r_g \sim U(0,1)$ and the behavioural parameters $\phi_p, \phi_g \in \mathbb{R}$. The behavioural parameters of the PSO method have a significant impact on the method's ability to optimize a given problem and will be studied in greater detail in chapter 5.

It is customary to impose limitations on the distance a particle can move in a single step [79]. This is done by bounding a particle's velocity $\vec{v}$ to the range of the search-space, so the particle can at most move from one boundary to the other in one step.

Adding the velocity to the particle's current position causes the particle to move to another position in the search-space, regardless of any improvement to its fitness. This is the second formula of PSO:

$$\vec{x} \leftarrow \vec{x} + \vec{v} \tag{2.4}$$

The PSO algorithm is shown in figure 2.4.

2.6 Performance Comparison

It is customary to report the empirical performance of an optimization method on a number of representative problems, although the NFL theorems warn us not to conclude too much from such results [2, Page 70]:

> ... comparisons reporting the performance of a particular algorithm with a particular parameter setting on a few sample problems are of limited utility. While such results do indicate behaviour on the narrow range of problems considered, one should be very wary of trying to generalize those results to the other problems.

Instead the NFL paper proposes to use performance measures derived from the probabilistic framework used to prove the NFL theorems [2, Section V.B], but those probabilistic measures are not used in this thesis for a number of reasons. Firstly, they are rather complicated and some even involving probabilities that are impossible to know for black-box fitness functions. Secondly, the measures are non-standard, thus making it difficult for other researchers and practitioners to relate the results of this thesis to their own work. Thirdly and perhaps most important, none of the measures seem to adequately rid themselves of the dependency on just a single or a few optimization problems without going into purely theoretical probability analysis. So while one can only agree with the insufficiencies of the customary way of measuring the performance of an optimization method, the probabilistic NFL framework does not seem to provide us with a viable alternative. So despite the pessimism of the NFL theorem, benchmark problems are often used in the literature when devising and testing new optimization methods.

2.6.1 Benchmark Problems

A suite of twelve benchmark problems is used here. The benchmark problems are widely used in the literature and taken from a larger suite collected by Yao et al. [80]. Some of the benchmark problems have been used for several decades, see e.g. [81] [82] [83] [84] [85]. The benchmark problems generalize to higher dimensionalities and they vary from uni-modal problems (one local optimum which is hence also the global optimum, e.g. the Sphere problem), to multi-modal problems (several local optima where the optimizer may get stuck, e.g. the Ackley problem), and from separable problems (the dimensions of the search-space are independent of each other in their influence on the fitness, e.g. the Sphere problem), to non-separable problems (the dimensions of the search-space may be intricately dependent on each other in their influence on the overall fitness, e.g. the Rosenbrock problem). The QuarticNoise problem contains stochastic noise, the Step problem is discontinuous, and the Penalized1 and 2 problems have constraints in the form of penalty functions. The formulae for computing the benchmark problems are shown in table 2.1. These problems all have an optimal fitness value of zero, although the QuarticNoise problem has noise added which makes it highly unlikely that a fitness value of zero can ever be

found. Graphical plots of the benchmark functions have been omitted because 2- and 3-dimensional plots are not representative of higher dimensionalities and experiments will be conducted for 30-dimensional problems.

Initialization, Search-Space & Displacement

Table 2.2 shows the initialization ranges used when optimizing these benchmark problems, as well as the search-space boundaries. The asymmetrical initialization ranges are chosen to further increase the difficulty of optimizing these benchmark problems. Initialization of an optimizing agent is done by making a uniformly distributed random sample in the designated range. Boundaries are a particularly simple form of constraints and are enforced by mere saturation to the boundary values in case an optimizer moves beyond them. Some optimization methods require boundaries to bounce off in order to perform well, while other optimization methods inherently never exceed their boundaries. But for the sake of simplicity and consistency, all experiments in this thesis are conducted with boundaries being enforced before fitness evaluation takes place.

As mentioned in the introductory chapter 1 the Stochastic GA (StGA) by Tu and Lu [42] was shown empirically to improve performance on a number of benchmark problems, but it was later discovered [43] that the StGA was in fact strongly attracted to the position zero, $\vec{0}$, and that just happened to be the location of the global optimum for the benchmark problems on which StGA was tested. To avoid this issue the global optima of the benchmark problems are displaced in the search-space. Displacement is also sometimes called shifting in the literature, see e.g. [86]. The displacement of the optimum for a fitness function f is achieved simply by using an analogous fitness function h:

$$h(\vec{x}) = f(\vec{x} - \delta)$$

where δ is the displacement value defined in table 2.2. These displacement values are chosen relative to the search-space boundaries for the benchmark problems and do not have a deep meaning apart from providing such displacement of the global optimum. Note, however, that some benchmark problems are displaced with negative values and others with positive values, which is intended to further diffuse any correlation. Also note that the Penalized1 and 2 problems are not displaced because they make use of a penalty function that assumes a certain location of the global optimum.

2.6.2 Measuring Optimization Performance

A basic performance criterion for an iterative optimization method is to measure the optimization results that can be achieved within a given amount of computation time. For most optimization problems of interest, the majority of the computation time is spent in the evaluation of the fitness function. The computation time is therefore generally measured as a number of evaluations of the fitness function, so as to keep the measure independent of the actual implementation and computer hardware.

Optimization performance will be depicted in a number of ways in this thesis. The end results of optimization are shown in terms of the mean and standard deviation of the fitness achieved, as well as the quartiles. This shows the center and dispersion of results in various ways. Graphical plots are also used to show the progress of optimization leading up to the end results.

For a series of values a_i of length N the mean or average $\bar{a}$ and the standard deviation s are given by:

$$\bar{a} = \frac{1}{N}\sum_{i=1}^{N} a_i, \qquad s = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(a_i - \bar{a})^2}$$

Concerning the method employed here for computing the quartiles, first sort the elements of a_i and call the sorted values b_i. If there is an odd number of elements then the median is the center element of the b_i series. If there is an even number of observations then the median is computed as the mean of the two center elements of the b_i series. The lower quartile $Q1$ is the element b_j with index $j = \lceil (1/4)(N+1) \rceil$, and the upper quartile $Q3$ is the element b_k at index $k = \lfloor (3/4)(N+1) \rfloor$. The best and worst results are also shown.

These measures taken together with the progress plots give a good view of the performance of an optimizer.

2.7 Experimental Results

This experiment will show the optimization performance that can be achieved using standard behavioural parameters:

- For the DE method a standard choice of allegedly good behavioural parameters is found in [16], which is a choice that also satisfies the theoretical conditions derived by Zaharie [30]. The hand-tuned DE parameters are:

  $$NP = 300,\ F = 0.5,\ CR = 0.9$$

  where NP denotes the number of agents in the population.

- For the PSO method, years of accumulated experience is reported in [87], suggesting the following parameters work well for that method:

  $$S = 50,\ \omega = 0.729,\ \phi_p = \phi_g = 1.49445 \tag{2.5}$$

  where S denotes the number of particles in the swarm.

- The LUS method is used with its standard choice of parameter: $\alpha = 1/3$

- The PS method does not have any behavioural parameters.

Using these behavioural parameters and conducting 50 optimization runs on each of the 12 benchmark problems in 30 dimensions and allowing 60,000 fitness

evaluations for each optimization run, gives the end results shown in table 2.3. The optimization progress is compared in terms of the mean fitness in figures 2.5 and 2.6 which are shown individually and in more detail in figures 2.7-2.14. The first thing to note is that the single-agent PS and LUS methods have rapid optimization progress in comparison to the multi-agent DE and PSO methods. Recall that PS and LUS were specifically designed to work well on simple optimization problems, which the results on e.g. the Sphere, Step, and QuarticNoise problems confirm. What is perhaps more interesting is that PS and LUS also work comparatively well on many of the harder problems, such as Ackley and Rastrigin where the DE and PSO methods seem to progress slowly or even stagnate. However, PS and LUS have somewhat irregular performance and are not always able to achieve good results, see for example their performance on the Ackley and Rosenbrock problems in figures 2.11-2.14. Recall that all benchmark problems have an optimal fitness value of zero, so from table 2.3 it can be seen that the LUS and PS methods are able to find near-optimal values for only a few of the problems within the computation time allowed. From their progress plots and our knowledge of the inner workings of the LUS and PS methods, which makes use of an exponentially decreasing sampling-range, it is likely that they either find a near-optimal solution early in the optimization run, or that they stagnate. Concerning the DE and PSO methods they are not even able to find near-optimal solutions for the simple Sphere problem. It is too early, though, to dismiss DE and PSO as poor optimizers, because the reason for this deficiency might be that DE and PSO have had their behavioural parameters hand-tuned for much longer optimization runs, say, when allowed a million or more fitness evaluations per optimization run. This is something that will be studied in more detail in chapters 4 and 5 when the behavioural parameters of DE and PSO will be tuned for a number of scenarios.

2.8 Summary

This chapter introduced two simple optimizers called PS and LUS, which work by locally sampling the surroundings of a single optimizing agent. Two popular multi-agent optimization methods from literature were also described, which work by having several optimizing agents working together in a competitive or cooperative manner. These were the DE and PSO methods.

On a benchmark suite of 12 optimization problems the PS and LUS methods were shown to have rapid, although sometimes also irregular optimization progress. The PS and LUS methods were able to find near-optimal solutions for some of the benchmark problems, while the DE and PSO methods were unable to achieve near-optimal solutions to even the simpler benchmark problems. The cause of this is perhaps that the hand-tuned behavioural parameters for DE and PSO were intended for much longer optimization runs. This is something that will be studied in detail in later chapters.

- Initialize $\vec{x}$ to a random position in the search-space:

$$\vec{x} \sim U\left(\vec{b}_{lo}, \vec{b}_{up}\right)$$

  where $\vec{b}_{lo}$ is the lower boundary of the search-space and $\vec{b}_{up}$ is the upper boundary.

- Set the initial sampling range $\vec{d}$ to cover the entire search-space:

$$\vec{d} \leftarrow \vec{b}_{up} - \vec{b}_{lo}$$

- Until a termination criterion is met, repeat the following:
  - Pick an index $R \in \{1, \cdots, n\}$ uniformly and randomly.
  - Let $\vec{y}$ be the potentially new position in the search-space, which is exactly the same as the current position $\vec{x}$, except for the R'th element y_R, which is found from the neighbourhood of x_R simply by adding d_R:

$$y_i = \begin{cases} x_i + d_i & , i = R \\ x_i & , \text{else} \end{cases} \tag{2.6}$$

  - If $(f(\vec{y}) < f(\vec{x}))$ then keep the new position:

$$\vec{x} \leftarrow \vec{y}$$

    Otherwise update the sampling-range and direction for the R'th dimension:

$$d_R \leftarrow -\frac{d_R}{2}$$

Figure 2.1: PS algorithm.

- Initialize $\vec{x}$ to a random position in the search-space:

$$\vec{x} \sim U\left(\vec{b}_{lo}, \vec{b}_{up}\right)$$

  where $\vec{b}_{lo}$ is the lower boundary of the search-space and $\vec{b}_{up}$ is the upper boundary.

- Set the initial sampling range $\vec{d}$ to cover the entire search-space:

$$\vec{d} \leftarrow \vec{b}_{up} - \vec{b}_{lo}$$

- Until a termination criterion is met, repeat the following:
  - Pick a random vector $\vec{a} \sim U\left(-\vec{d}, \vec{d}\right)$
  - Add this to the current position $\vec{x}$, to create the new potential position $\vec{y}$:

$$\vec{y} = \vec{x} + \vec{a}$$

  - If $(f(\vec{y}) < f(\vec{x}))$ then update the position:

$$\vec{x} \leftarrow \vec{y}$$

    Otherwise decrease the sampling-range by the factor q from Eq.(2.1):

$$\vec{d} \leftarrow q \cdot \vec{d}$$

Figure 2.2: LUS algorithm.

- Initialize the agents with random positions in the search-space.
- Until a termination criterion is met, repeat the following:
  - For each agent $\vec{x}$ in the population do the following:
    * Pick three agents $\vec{a}$, $\vec{b}$ and $\vec{c}$ at random, they must be distinct from each other as well as from agent $\vec{x}$.
    * Pick a random index $R \in \{1, \cdots, n\}$, where the highest possible value n, is the dimensionality of the problem to be optimized.
    * Compute the agent's potentially new position $\vec{y} = [y_1, \cdots, y_n]$, by iterating over each $i \in \{1, \cdots, n\}$ as follows:
      · Pick $r_i \sim U(0, 1)$ for use in a stochastic choice next.
      · Compute the i'th element of the potentially new position $\vec{y}$, using Eq.(2.2) from above:

$$y_i = \begin{cases} a_i + F(b_i - c_i) & \text{, if } (i = R) \text{ or } (r_i < CR) \\ x_i & \text{, else} \end{cases}$$

        Where the user-defined behavioural parameters are the differential weight F and the crossover probability CR.
    * If $(f(\vec{y}) < f(\vec{x}))$ then update the agent's position:

$$\vec{x} \leftarrow \vec{y}$$

Figure 2.3: DE algorithm.

- Initialize the particles with random velocities and random positions in the search-space.
- Until a termination criterion is met, repeat the following:
  - For each particle in the swarm do the following:
    * Pick two random numbers: r_p, $r_g \sim U(0,1)$
    * Update the particle's velocity $\vec{v}$ as follows:

$$\vec{v} \leftarrow \omega\vec{v} + \phi_p r_p(\vec{p} - \vec{x}) + \phi_g r_g(\vec{g} - \vec{x})$$

      Where $\vec{g}$ is the swarm's best known position, $\vec{p}$ is the particle's own best known position, and ω, ϕ_p, and ϕ_g are user-defined behavioural parameters.
    * Move the particle to its new position by adding its velocity:

$$\vec{x} \leftarrow \vec{x} + \vec{v}$$

    * If $(f(\vec{x}) < f(\vec{p}))$ then update the particle's best known position:

$$\vec{p} \leftarrow \vec{x}$$

    * If $(f(\vec{x}) < f(\vec{g}))$ then update the swarm's best known position:

$$\vec{g} \leftarrow \vec{x}$$

Figure 2.4: PSO algorithm.

Ackley	$f(\vec{x}) = e + 20 - 20 \cdot \exp\left(-0.2 \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)$
Griewank	$f(\vec{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)$
Penalized1	$f(\vec{x}) = \frac{\pi}{n}\Big(10 \cdot \sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i-1)^2 \cdot \big(1 + 10 \cdot \sin^2(\pi y_{i+1})\big) + (y_n - 1)^2\Big) + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$ $y_i = 1 + (x_i + 1)/4$ $u(x_i, a, k, m) = \begin{cases} k(-x_i - a)^m & , x_i < -a \\ 0 & , -a \le x_i \le a \\ k(x_i - a)^m & , x_i > a \end{cases}$
Penalized2	$f(\vec{x}) = 0.1\Big(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i-1)^2 \cdot \big(1 + \sin^2(3\pi x_{i+1})\big) + (x_n - 1)^2 \cdot \big(1 + \sin^2(2\pi x_n)\big)\Big) + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$, with $u(\cdot)$ from above.
QuarticNoise	$f(\vec{x}) = \sum_{i=1}^{n}(i \cdot x_i^4 + r_i),\ r_i \sim U(0,1)$
Rastrigin	$f(\vec{x}) = \sum_{i=1}^{n}\big(x_i^2 + 10 - 10 \cdot \cos(2\pi x_i)\big)$
Rosenbrock	$f(\vec{x}) = \sum_{i=1}^{n-1}\big(100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2\big)$
Schwefel1-2	$f(\vec{x}) = \sum_{i=1}^{n}\Big(\sum_{j=1}^{i} x_j\Big)^2$
Schwefel2-21	$f(\vec{x}) = \max\{\lvert x_i\rvert : i \in \{1, \cdots, n\}\}$
Schwefel2-22	$f(\vec{x}) = \sum_{i=1}^{n}\lvert x_i\rvert + \prod_{i=1}^{n}\lvert x_i\rvert$
Sphere	$f(\vec{x}) = \sum_{i=1}^{n} x_i^2$
Step	$f(\vec{x}) = \sum_{i=1}^{n}\left(\lfloor x_i + 0.5\rfloor\right)^2$

Table 2.1: Benchmark problems used in this study.

Problem	Initialization	Search-Space	Displacement δ
Ackley	$[15, 30]$	$[-30, 30]$	-7.5
Griewank	$[300, 600]$	$[-600, 600]$	-150
Penalized1	$[5, 50]$	$[-50, 50]$	0
Penalized2	$[5, 50]$	$[-50, 50]$	0
QuarticNoise	$[0.64, 1.28]$	$[-1.28, 1.28]$	-0.32
Rastrigin	$[2.56, 5.12]$	$[-5.12, 5.12]$	1.28
Rosenbrock	$[15, 30]$	$[-100, 100]$	25
Schwefel1-2	$[50, 100]$	$[-100, 100]$	-25
Schwefel2-21	$[50, 100]$	$[-100, 100]$	-25
Schwefel2-22	$[5, 10]$	$[-10, 10]$	-2.5
Sphere	$[50, 100]$	$[-100, 100]$	25
Step	$[50, 100]$	$[-100, 100]$	25

Table 2.2: Initialization ranges, search-space boundaries, and displacement values δ for the benchmark problems. These values are used for all dimensions.

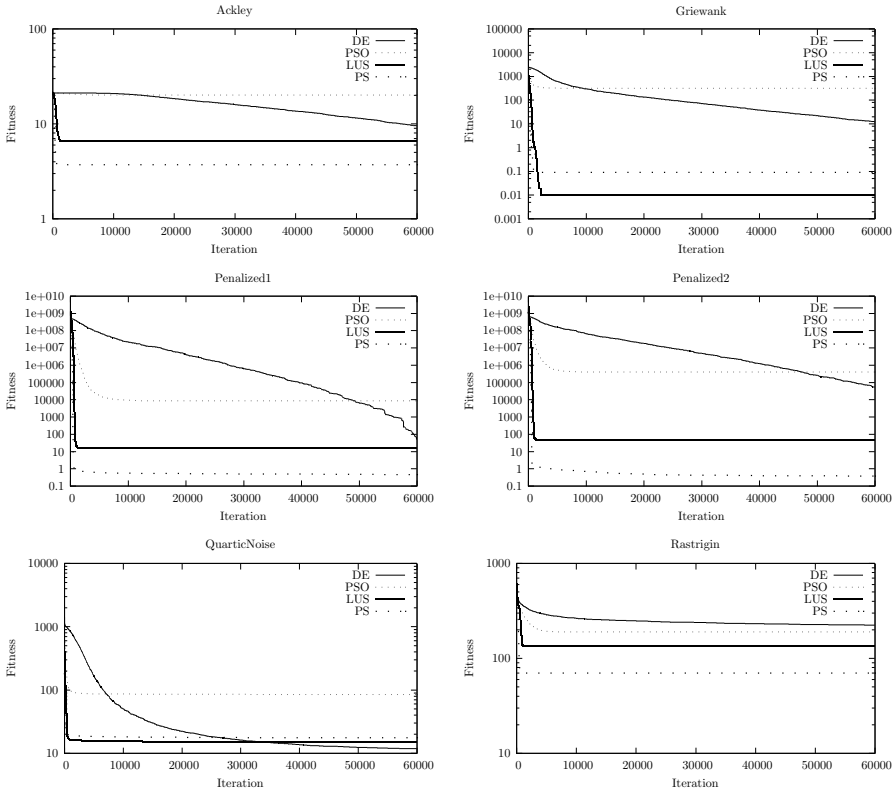

Figure 2.5: Comparison of optimization progress using **hand-tuned** behavioural parameters. Plots show the mean fitness achieved over 50 optimization runs.

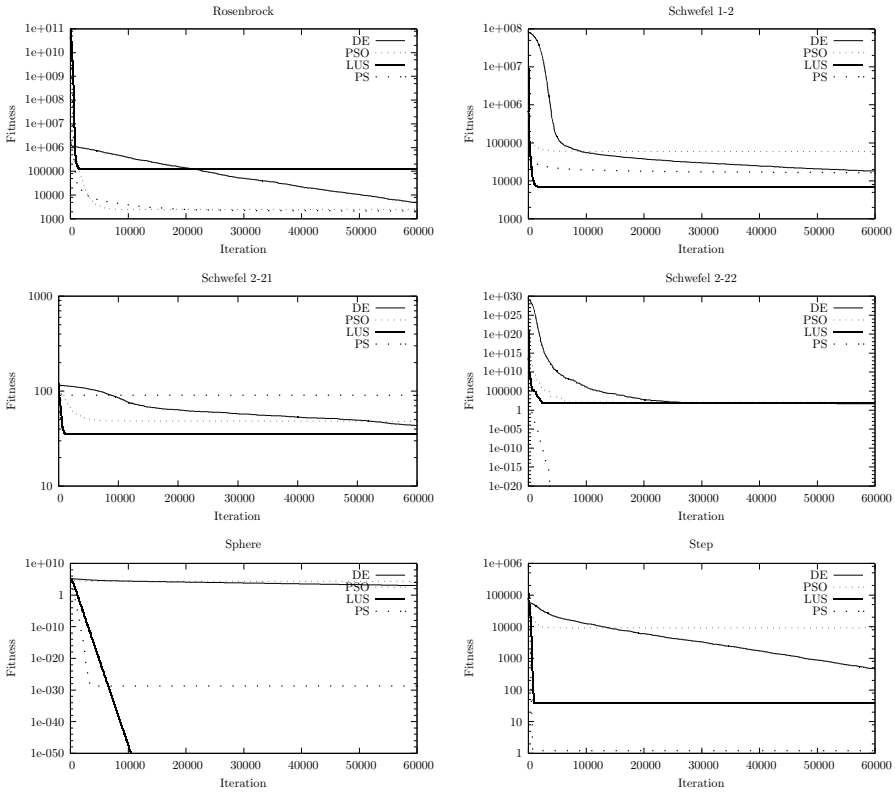

Figure 2.6: Comparison of optimization progress using **hand-tuned** behavioural parameters. Plots show the mean fitness achieved over 50 optimization runs.

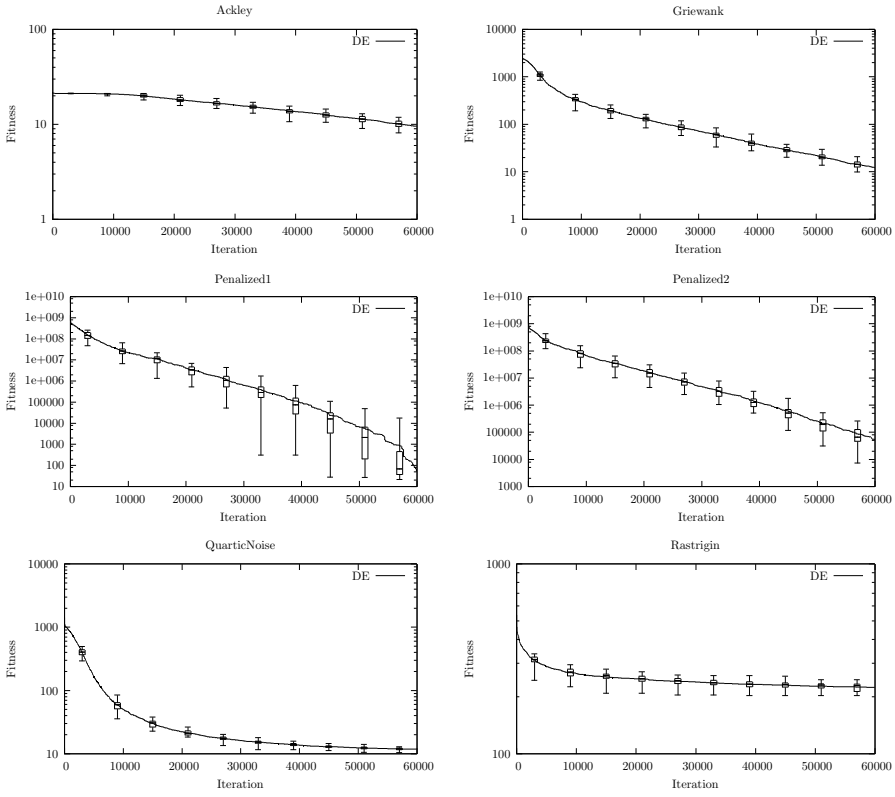

Figure 2.7: **DE/rand/1/bin** optimization progress using the **hand-tuned** behavioural parameters $NP = 300$, $CR = 0.9$, $F = 0.5$. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

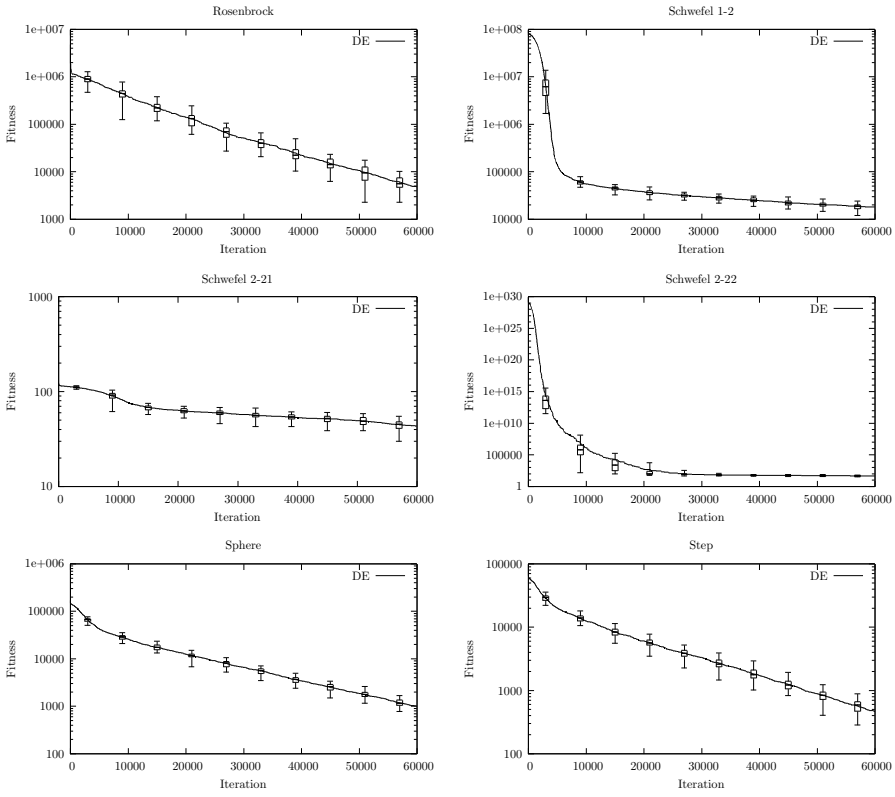

Figure 2.8: **DE/rand/1/bin** optimization progress using the **hand-tuned** behavioural parameters $NP = 300$, $CR = 0.9$, $F = 0.5$. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

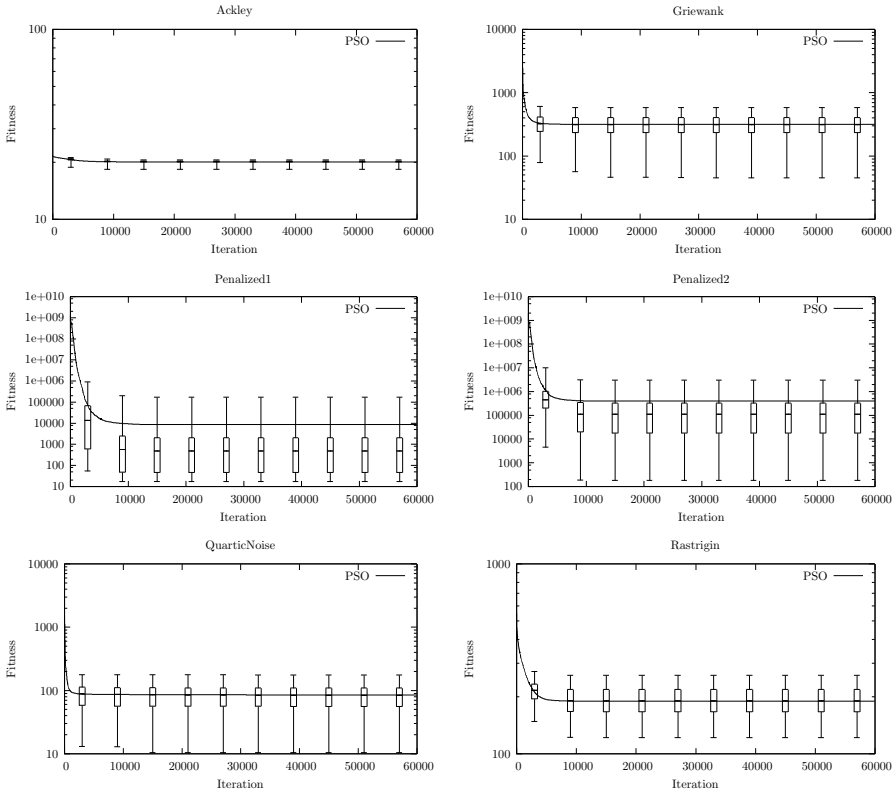

Figure 2.9: **PSO** optimization progress using the **hand-tuned** behavioural parameters $S = 50$, $\omega = 0.729$, $\phi_p = \phi_g = 1.49445$. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

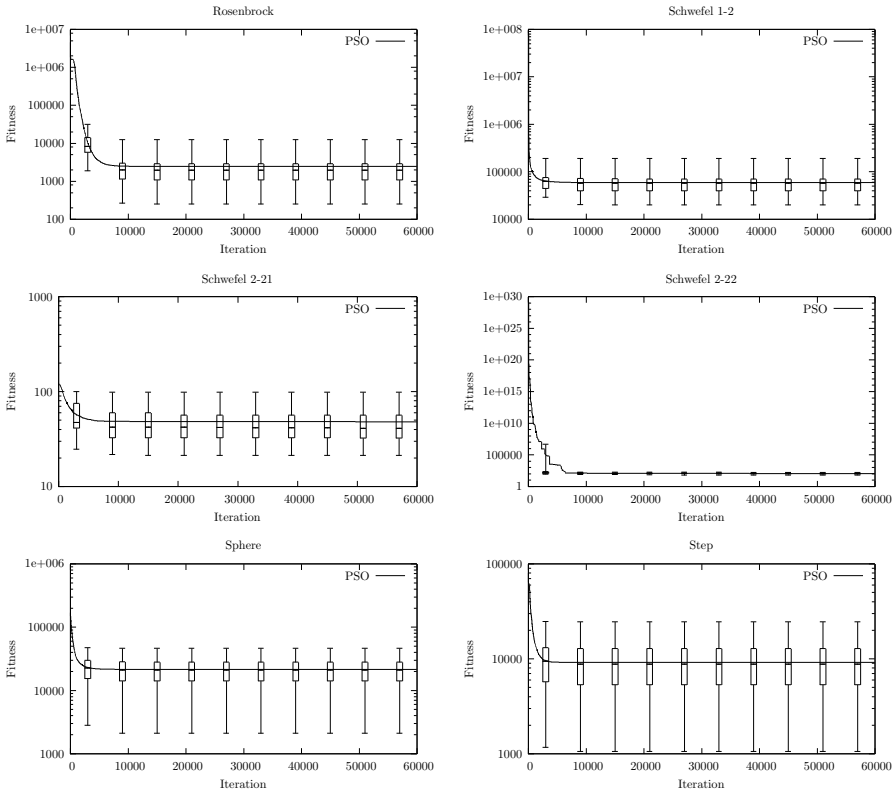

Figure 2.10: **PSO** optimization progress using the **hand-tuned** behavioural parameters $S = 50$, $\omega = 0.729$, $\phi_p = \phi_g = 1.49445$. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

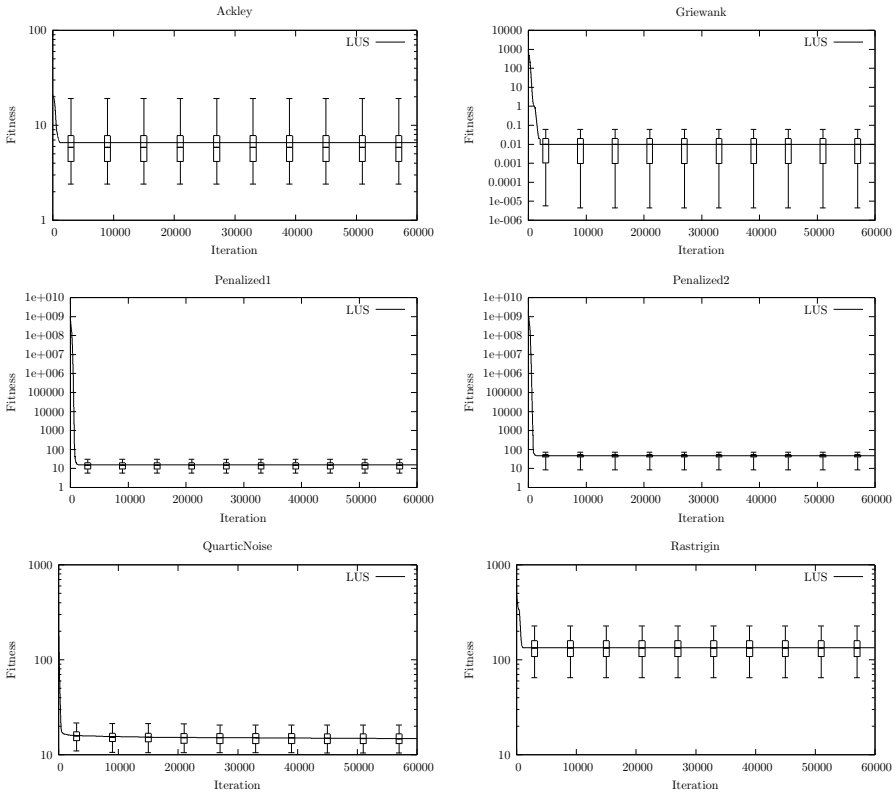

Figure 2.11: **LUS** optimization progress using its standard behavioural parameter $\alpha = 1/3$. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

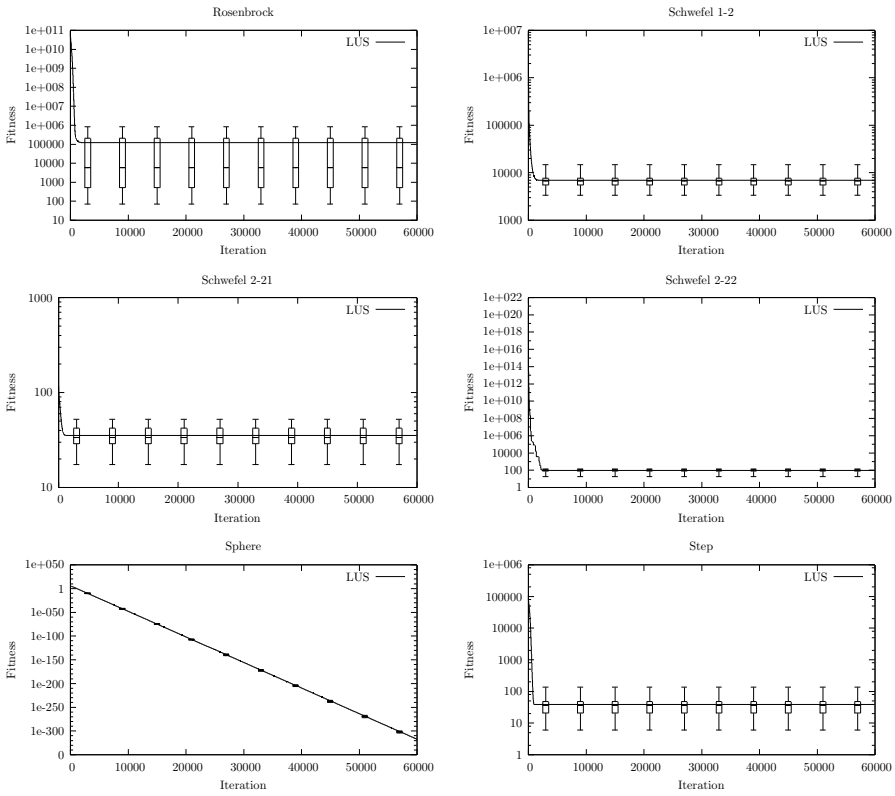

Figure 2.12: **LUS** optimization progress using its standard behavioural parameter $\alpha = 1/3$. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

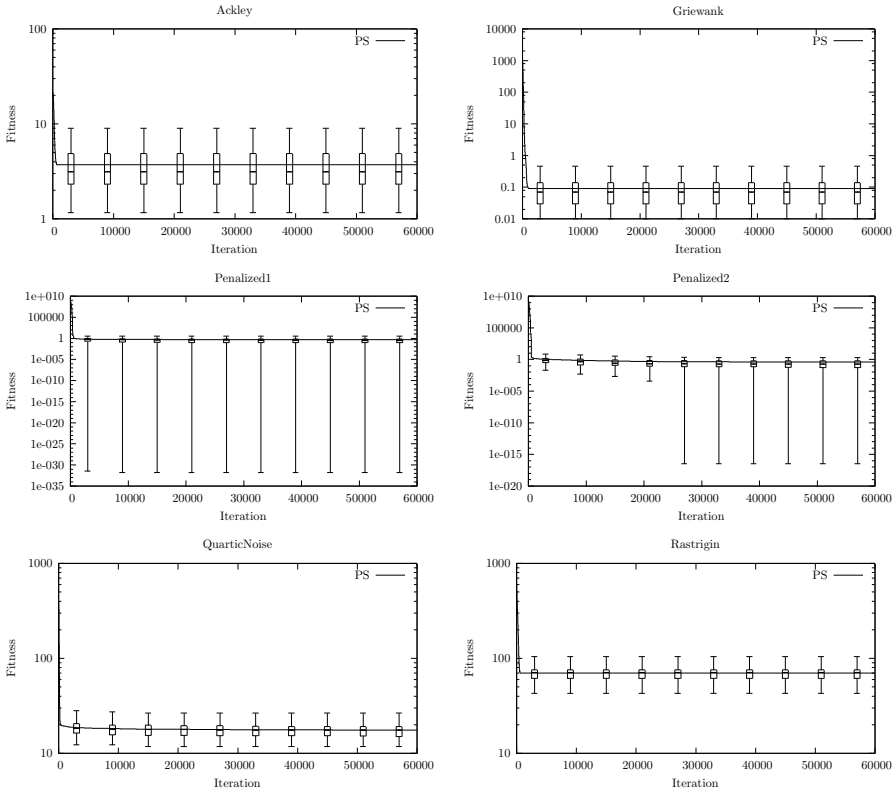

Figure 2.13: **PS** optimization progress. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

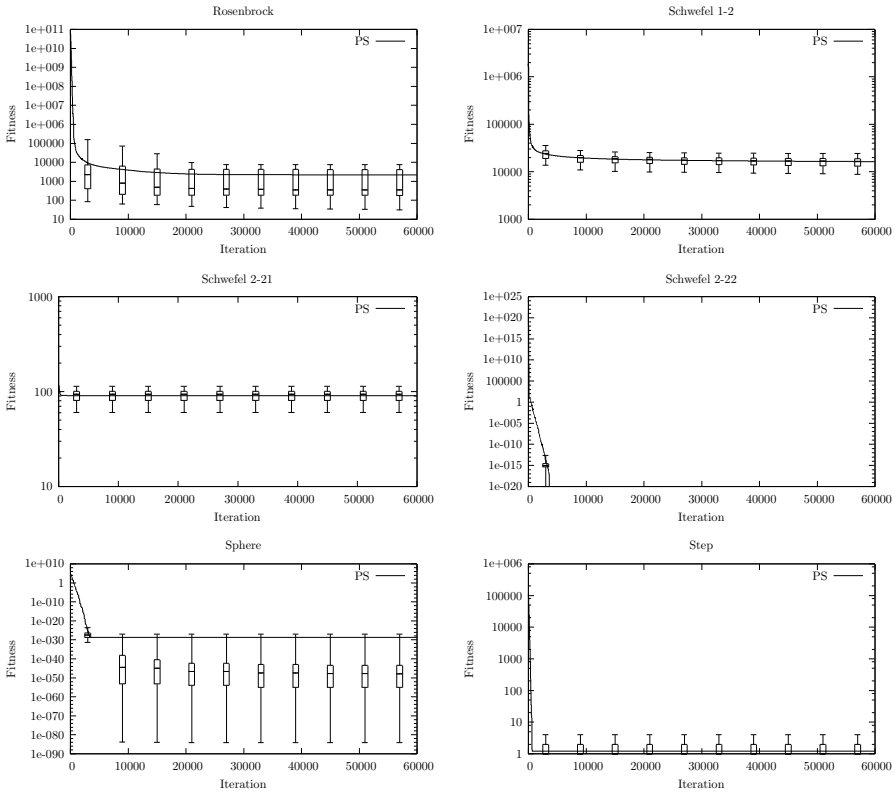

Figure 2.14: **PS** optimization progress. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization. Note that a fitness value of zero is reached for the Schwefel 2-22 and Step problems, which cannot be plotted on a logarithmic scale.

	Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
DE	Ackley	9.55	0.74	7.56	9.06	9.58	10.13	11.13
	Griewank	12.32	1.86	8.76	11.05	11.95	13.56	17.21
	Penalized1	53.98	60.98	19.31	26.24	36.59	45.16	340.07
	Penalized2	54440	46576	2104	22245	40992	64051	204258
	QuarticNoise	11.83	0.59	10.49	11.52	11.86	12.3	12.95
	Rastrigin	223.58	10.81	202.23	211.99	226.44	232.28	239.42
	Rosenbrock	4833	1380	2300	3781	4759	5476	7705
	Schwefel1-2	18091	2207	12075	16632	18698	19609	22646
	Schwefel2-21	43.43	4.88	29.95	39.33	43.97	47.07	53.47
	Schwefel2-22	47.84	5.21	34.75	44.83	47.65	50.85	63.64
	Sphere	1002	175.23	618.91	855.34	1009	1143	1403
	Step	465.86	109.38	272	371	471	557	766
PSO	Ackley	20.06	0.43	18.3	20.09	20.13	20.22	20.55
	Griewank	316.42	121.71	45.25	234.09	313.82	403.39	580.54
	Penalized1	8646	27811	17.02	47.07	479.92	2068	170234
	Penalized2	398028	696938	182.72	18140	109865	325680	3.06e+6
	QuarticNoise	85.05	41.31	10.47	55.94	84.26	109.39	175.27
	Rastrigin	189.42	33.58	121.82	166.73	189.75	218.23	258.6
	Rosenbrock	2473	2155	253.71	1083	1952	2910	12426
	Schwefel1-2	59270	27586	20042	39751	57992	70949	191536
	Schwefel2-21	48.05	22.07	21.3	32.53	41.06	56.69	98.46
	Schwefel2-22	112.28	24.82	64.53	94.14	115.29	129.49	169.07
	Sphere	21537	10375	2118	14225	21095	28210	46502
	Step	9194	4814	1055	5357	8803	12810	24547
LUS	Ackley	6.57	3.18	2.41	4.17	5.86	7.78	19.16
	Griewank	0.01	0.01	4.43e-6	9.78e-4	9.91e-3	0.02	0.06
	Penalized1	15.34	6.2	5.74	9.42	15.23	20.17	30.51
	Penalized2	47.21	11.43	8.48	41.09	46.36	54.13	72.29
	QuarticNoise	14.82	2.53	10.46	13.15	14.73	16.61	20.56
	Rastrigin	133.52	33.91	64.67	108.45	133.32	159.19	226.85
	Rosenbrock	122074	205881	71.4	514.47	5969	204828	839108
	Schwefel1-2	6920	2249	3374	5521	6801	7671	14808
	Schwefel2-21	35.3	8.21	17.49	29.02	33.62	42.11	52.17
	Schwefel2-22	92.17	27.45	17.74	83.67	98.52	111.57	136.31
	Sphere	7.09e-318	0	2.09e-321	1.27e-319	9.87e-319	2.96e-318	1.53e-316
	Step	38.36	23.43	6	21	36	48	138
PS	Ackley	3.7	1.8	1.16	2.32	3.13	4.88	8.99
	Griewank	0.09	0.09	0	0.03	0.07	0.14	0.46
	Penalized1	0.47	0.54	1.57e-32	0.1	0.3	0.65	3.14
	Penalized2	0.38	0.47	3.49e-17	0.05	0.18	0.63	2.05
	QuarticNoise	17.55	3.3	11.79	14.97	17.55	19.1	26.47
	Rastrigin	69.85	11.92	42.78	61.69	71.14	75.62	104.47
	Rosenbrock	2150	2769	30.94	182.27	351.58	4174	7705
	Schwefel1-2	16397	3836	8826	13199	16579	18768	24278
	Schwefel2-21	90.3	13.41	60.17	80.79	93.87	100.77	113.32
	Schwefel2-22	0	0	0	0	0	0	0
	Sphere	1.83e-29	1.28e-28	7.88e-85	8.43e-56	1.27e-48	4.48e-44	9.14e-28
	Step	1.22	1.04	0	0	1	2	4

Table 2.3: End results on benchmark problems of 30 dimensions each using **hand-tuned** behavioural parameters. Results obtained over 50 optimization runs where the number of fitness evaluations for each run is 60,000.

Chapter 3

Meta-Optimization

3.1 Introduction

It has been recognized since the inception of DE that different choices of behavioural parameters cause it to perform worse or better on particular problems, and that the selection of good parameters is a challenging art, see for example Storn et al. [17] [25], Liu and Lampinen [26], and Zaharie [30]. Similarly, selecting good behavioural parameters for PSO is also challenging, see e.g. Shi and Eberhart [22] [23], Carlisle and Dozier [24], van den Bergh [27], Trelea [28], and Clerc and Kennedy [29].

This chapter considers the task of finding a good choice of behavioural parameters for an optimizer, as an optimization problem in its own right and dubs this *Meta-Optimization*. In other words, the idea is to have a black-box optimization method act as an overlaying meta-optimizer, trying to find good performing behavioural parameters for another optimization method, which in turn is used to optimize a number of problems. This tuning of an optimizer's parameters is done in an offline manner and the parameters can therefore be used by other researchers and practitioners without modifying their own implementations of the optimization algorithm, but merely replace the behavioural parameters. The overall concept of meta-optimization is depicted in figure 3.1.

The chapter is organized as follows: First an overview of related work is given. Then the performance measure of an optimization method is formalized along with discussions and literature surveys. Experiments are then conducted to establish which optimization method should be used as the meta-optimizer.

3.1.1 Related Work

The concept of meta-optimization is not new and is also known in the literature as Meta-Evolution, Super-Optimization, Automated Parameter Calibration, Hyper-Heuristics etc. [31] [88] [89] [32] [90] [91] [92]. In fact, meta-optimization is reported to have been used back in the late 1970's by Mercer and Sampson [93] for finding optimal parameter settings of a GA. Another early

example of meta-optimizing discrete parameters of a GA is due to Grefenstette [31], while meta-optimizing both the behavioural parameters and the GA operators are studied by Bäck [88]. Results with meta-optimizing GA parameters are also reported by Keane [94]. But meta-optimization is very time-consuming and it was not until recently that computers had enough processing power to conduct somewhat more realistic experiments in meta-optimization, see for example the experiments by Meissner et al. [32], and the comparison of various meta-optimization techniques by Smit and Eiben [33].

The approaches to meta-optimization found in the literature differ in a number of ways from the work in this thesis, including the following:

- Previous approaches to meta-optimization are more complicated, some even requiring special variants of the optimizers that are being used as the overlaying meta-optimizer, see for example [88]. The present work on the other hand, gives a simple framework in which any black-box optimization method can be used as the overlaying meta-optimizer to find the behavioural parameters of any other optimization method.

- Previous approaches to meta-optimization do not properly address the issue of reducing computational time without either parallelizing the implementation or reducing the quality of the results. Even with the speed of modern computers, full-blown experiments in meta-optimization are still very time-consuming. The time expenditure of meta-optimization is lowered significantly in the present work by applying a simple programming expedient dubbed *Preemptive Fitness Evaluation*.

- Previous approaches do not use suitable optimization methods as the overlaying meta-optimizer. The early work all used a GA as the meta-optimizer, while recent work also use recent black-box optimization methods, such as a PSO [32] or a state-of-the-art ES variant [33]. But due to the computational cost involved in meta-optimization only few iterations of the meta-optimizer can be performed in a reasonable amount of time, yet GA, PSO, etc. typically require thousands of fitness evaluations to achieve satisfactory goals. In the work presented here the LUS method from chapter 2 is used as the overlaying meta-optimizer as it often performs very well, yet only requires a very small number of fitness evaluations. Furthermore, the LUS method is simpler to describe and implement, making the present work more applicable.

Another recent approach to meta-optimization is called REVAC and is due to Nannen and Eiben [90], where in addition to tuning the behavioural parameters a measure of relevance for individual parameters is also made. This allows the meta-optimizer to focus its effort on the parameters that are most likely to cause an improvement in the performance of the optimizer, something which could also prove useful in later chapters of this thesis, where the less relevant parameters are being eliminated. The REVAC approach also gives a way of lowering the costly time usage involved in meta-optimization [95]. However,

the REVAC approach is still significantly more complicated than the approach to meta-optimization presented here, and it has a significant drawback in that it treats the behavioural parameters individually and thus ignores how they influence each other in a non-linear way.

A statistical approach to meta-optimization is due to François and Lavergne [96], where a statistical model is built from numerical experiments on how the behavioural parameters of an optimization method influence its performance. The technique is claimed to have the ability to tune behavioural parameters so as to generalize well to a class of similar optimization problems, without incurring additional computational cost during the parameter tuning process. But the technique and its presentation is mathematical and complicated, thus lacking the immediacy and grace that made black-box optimization methods attractive for the average practitioner; something which is the aim of the meta-optimization technique presented here.

In short, the approach to meta-optimization taken here appears to be significantly simpler, less time-consuming, yields better results, and hence seems to be more applicable than the approaches taken previously in the literature.

3.2 Multi-Objective Optimization

The crux of automatically finding good behavioural parameters for an optimization method is to define an appropriate meta-fitness measure, which can be made the subject of black-box optimization. A simple way of judging the performance of an optimization method and a specific choice of behavioural parameters, is to consider the fitness results obtained when optimizing a number of different benchmark problems for a number of iterations and a number of times. Using several benchmark functions to judge the quality of an optimization method is done in an effort to make the behavioural parameters work well on a broader range of problems. But this results in many different values to be compared, some of which are better and some worse – and until now, we have relied on flexible human evaluation to summarize all these benchmark measures, to decide which optimization method was actually better than the other. In essence, this is what is known as *multi-objective* optimization for the meta-fitness layer.

3.2.1 Multi-Objective Fitness Function

Multi-objective optimization is now formalized in general terms. Optimization problems having N objectives are generally modelled by having the fitness function map a position in the search-space $x \in X$ to a vector of N real-valued fitness measures, each corresponding to an optimization objective:

$$f : X \to \mathbb{R}^N$$

This could also be written as a set of N fitness functions, with each f_i mapping from X to $\mathbb{R}$ as usual.

3.2.2 Pareto Optimality

The difficulty with optimizing multiple objectives simultaneously is that the objectives may be conflicting or contradictory. So when the optimizer improves the fitness of one objective, it may worsen the fitness of another objective. To describe this more formally, the mathematical definition of optimality from chapter 1 must first be extended to the case of multiple objectives.

Domination

A position in the search-space $y \in X$ is said to *dominate* another position x, if y rates at least as well as x on all objectives, and better than x on at least one of the objectives. This domination can be denoted by $y <_p x$ and defined as:

$$y <_p x \quad \Leftrightarrow \quad \forall i : \ f_i(y) \leq f_i(x) \quad \wedge \quad \exists i : \ f_i(y) < f_i(x) \tag{3.1}$$

A position in the search-space x is then said to be *Pareto* optimal if there does not exist any $y \in X$ that dominates x. In other words, x is Pareto optimal if it is non-dominated, which can be written as:

$$x \in X \text{ is Pareto optimal} \quad \Leftrightarrow \quad \nexists \, y \in X : \ y <_p x$$

An early text on Pareto optimality is [97].

Pareto Decisions

The above definition of Pareto optimality can be used in different ways for multi-objective optimization. For example, optimization methods which do not require an actual fitness value, but merely need to decide which of the two candidate solutions x or y is better in order to guide their optimization, can use Eq.(3.1) directly in place of their usual fitness comparison. This is used by Horn et al. [98] for a slightly more sophisticated version of *tournament selection* in a GA.

Pareto Ranking

Making simple multi-objective decisions based on Eq.(3.1) will not work for optimization methods which require actual fitness values. However, the definition of Pareto optimality from Eq.(3.1) can still be used to assign singleton fitness values to candidate solutions and summarizing their mutual rankings on different objectives. But this only works for multi-agent optimization methods, as the idea is to rank the agents in a population according to how many of the other agents they are dominated by. The agents with the best rank are thus dominated by the least agents and are therefore considered better in the Pareto sense. This was originally suggested by Goldberg [12] and the ranking algorithm works by iteratively tagging non-dominated agents from a population. The rank of an agent is hence determined by the number of iterations that have been performed so far, and these ranks are used as fitness values during optimization. A slightly

different approach to this ranking approach is due to Fonseca and Fleming [99], and yet another variant is the *Non-Dominated Sorting GA* (NSGA) by Srinivas and Deb [100]. A more recent but also more complicated improvement to the latter, called NSGA-II, is due to Deb et al. [101], which among other things alleviates part of the costly overhead associated with identifying and sorting the non-dominated agents. The underlying idea of the NSGA-II is combined with DE to form the NSDE by Iorio and Li [102], which is reported to improve the performance on rotated optimization problems; which is a more difficult class of problems because the individual dimensions are non-separable.

Suitability To Meta-Optimization

The above techniques for using the Pareto definition directly are, however, not suited for meta-optimization. First, because the techniques are more complicated than what is required and, second, because such a precise definition of multi-objective optimality is probably not desirable in meta-optimization. Here, it is more important that the behavioural parameters discovered for an optimization method also generalize well to other problems and that the performance is good in an overall manner.

3.2.3 Weighted Sum

One of the simplest approaches to multi-objective optimization is to combine the individual objectives into a single fitness measure by weighting their sum. This technique is mentioned in [103, Chapter C4.5] as having been popular for many years. The weighted-sum approach also works well for meta-optimization, as it is merely the overall performance that matters, which is closer to how we as humans have evaluated the performance of optimization methods using benchmark results.

Formally, to compute the weighted sum of multiple objectives, let h be the new fitness function that is to be optimized, defined as:

$$h(x) = \sum_{i=1}^{N} w_i \cdot f_i(x) \tag{3.2}$$

with weights $w_i > 0$ and assuming that all objectives f_i are to be minimized.

3.3 Meta-Fitness Algorithm

This section outlines the computation of the meta-fitness measure that is used in meta-optimization for rating the performance of an optimizer with a given choice of behavioural parameters. As argued above, a suitable technique for combining a number of benchmark performances into a single meta-fitness measure that can be made the subject of meta-optimization, is a weighted sum of the optimization results on the individual benchmark problems.

Evaluating the performance of an optimization method and a particular choice of behavioural parameters then becomes a matter of iterating through the different problems, performing a number of optimization runs on each of them in turn, and adding up the resulting fitness values. This algorithm can be summarized roughly as in figure 3.2, with N being the number of problems the optimizer's behavioural parameters are being tuned for, enumerated as f_i, and M being the number of optimization runs to perform on each of these problems. The resulting meta-fitness measure is the variable s, which is being accumulated during the iterations of the algorithm.

3.3.1 Weights & Normalization

Since the benchmark problems can take on very different ranges of fitness values it seems likely that the meta-fitness measure will be dominated by the optimization results on one or more of the benchmark problems. For this reason the meta-fitness algorithm in figure 3.2 makes use of weights w_i to change the bias of optimization performance on the problems f_i. Selecting these weights is difficult, however. The immediate idea would be to choose each weight as the inverse of the maximum fitness value of that benchmark problem, that is, $w_i = 1/\max(f_i)$, so as to normalize the contribution of each benchmark problem to the overall meta-fitness sum. But this actually just reverses the bias as can be demonstrated with a small example.

Consider the Sphere and Rosenbrock problems in 30 dimensions each, and sample their search-spaces at random 6,000 times each (for in general, $\max(f_i)$ might not be known.) The mean fitness obtained for the Sphere problem is roughly 44000 and the mean fitness for Rosenbrock is roughly 1e+10. Clearly, using their fitness values directly in the meta-fitness sum would mean that Rosenbrock would dominate the Sphere problem. Both problems have an optimal fitness value of zero and the Sphere problem has acceptable solutions close to that because the problem is so simple, while acceptable solutions for Rosenbrock can have much larger fitness due to the difficulty in optimizing that problem. But an unacceptable result for Rosenbrock would surely be a fitness larger than, say, 1000. So if an optimizer were to achieve a fitness of, say, 0.01 for the Sphere problem and a fitness of 1000 for Rosenbrock, then their normalized contributions to the meta-fitness sum would be 2.27e-7 and 1e-7, respectively. This would mean that the result on Rosenbrock which was in fact unacceptable now rates similar to a result for the Sphere problem which was acceptable. That is, the bias has reversed and Sphere is given most weight and Rosenbrock least weight in making up the meta-fitness measure.

There does not seem to be a simple solution to this dilemma. Fortunately it turns out to be unimportant and all weights can usually be set to one, $w_i = 1$. That it actually works will become self-evident shortly and in the many experiments of chapters 4 and 5 where it is being used. Chapter 4 also contains experiments with choosing the weights.

3.3.2 Meta-Fitness Landscapes

To make a 3-dimensional plot showing what a meta-fitness landscape looks like, an optimization method having only two behavioural parameters is needed. Consider the DE/rand/1/bin optimizer having a fixed behavioural parameter $CR = 0.9$, which was supposedly a good choice according to recommendations in literature, see section 2.7. The other two behavioural parameters for the DE optimizer, NP and F, are then varied so as to compute a grid of meta-fitness values which can be plotted. The boundaries for the parameters are chosen so as to cover the ranges commonly used in the literature, see e.g. [16] [17] [26].

Figures 3.5 and 3.6 show the meta-fitness landscape when DE is used with different choices of behavioural parameters to optimize the Sphere and Rosenbrock benchmark problems and being allowed either 6,000 or 60,000 iterations per optimization run. Recall that smaller meta-fitness values mean better optimization performance because Sphere and Rosenbrock are minimization problems. The first thing to observe is how regular and simple these meta-fitness landscapes are. This explains why researchers and practitioners have had some degree of success in tuning the DE parameters by hand, and it also suggests that a simple optimization method can be used as the overlaying meta-optimizer to automate the parameter tuning process. Another interesting observation is that the fewer optimization iterations DE is allowed when optimizing the Sphere and Rosenbrock problems, the harder it becomes to select good behavioural parameters. This can be seen by the narrowing valley in the meta-fitness landscape when few optimization iterations are allowed. Note, for instance, that the population size NP can be chosen much larger when more optimization iterations are allowed, which perhaps makes intuitive sense. The region of the meta-fitness landscape holding the best choices of behavioural parameters seems to move slightly when more optimization iterations are allowed, but a choice of behavioural parameters that appears to perform well for both scenarios is $NP \simeq 40$, $F \simeq 0.6$, and $CR = 0.9$.

Figures 3.7 and 3.8 show the meta-fitness landscapes when using all 12 benchmark problems instead of just the Sphere and Rosenbrock problems. Due to the vast range of meta-fitness values plots are shown in figures 3.9 and 3.10 with the meta-fitness capped at 1e+15, so as to better show the interesting valley of good performing parameter combinations. These valleys can be seen to be quite similar in shape and placement to before, with the best performing parameters situated around $NP \simeq 50$ and $F \simeq 0.6$.

The similarity of the meta-fitness landscapes suggests that the Sphere and Rosenbrock problems may be representative for the entire suite of 12 benchmark problems, in the sense that the best performing DE parameters for use on the Sphere and Rosenbrock problems are similar to the best performing parameters for use on all 12 benchmark problems. Though, it should be kept in mind that the CR parameter is being held fixed and the meta-fitness landscapes may be dramatically different when CR is allowed to vary as well. It will also be likely that other optimization methods, and indeed other DE variants, will have different performance correlations amongst the benchmark problems.

So it cannot be concluded from just these experiments that the Sphere and Rosenbrock problems are representative for the entire benchmark suite.

Figures 3.11 and 3.12 show the optimization progress of using DE/rand/1/bin with behavioural parameters chosen from the meta-fitness landscapes, namely $NP = 50$, $F = 0.6$, and $CR = 0.9$. These are compared to the results of using hand-tuned behavioural parameters and show a great improvement. This would suggest that the meta-fitness measure is suitable for use in tuning the behavioural parameters of an optimization method.

3.3.3 Time Usage

Finding good choices of behavioural parameters for an optimization method was done by exhaustive search in the above, as a grid of meta-fitness values was computed and the best parameters could then be picked out. Although it would not be possible to visualize the meta-fitness landscapes for optimizers having more than two behavioural parameters, such a grid of meta-fitness values could still be computed and exhaustively searched. What prohibits this is the computation time required. Table 3.1 shows the time usage for computing the meta-fitness landscapes from above on an Intel Pentium M 1.5 GHz laptop computer. In chapter 4 an optimizer having 9 behavioural parameters is being studied, where a grid of similar resolution would take more than two billion years to compute. This is the Curse of Dimensionality for the meta-fitness search-space. Clearly, it is not tractable to perform an exhaustive search for good behavioural parameters of an optimizer, just as it is not tractable to perform an exhaustive search for the optimum of any other non-trivial problem.

3.4 Preemptive Fitness Evaluation

Having defined a meta-fitness measure for use in meta-optimization, this section now describes a technique for greatly reducing the time expenditure of computing the meta-fitness measure, when it is being optimized by certain kinds of methods. The time-saving technique can be used for many other optimization problems and will therefore be described in general terms.

The technique is dubbed *Preemptive Fitness Evaluation* because it works by preemptively aborting a fitness (or meta-fitness) evaluation, once the fitness becomes worse than that needed for it to be accepted as an improvement by the optimizer (or meta-optimizer), and the fitness is known not to improve for the rest of the evaluation.

Although the technique has been used by researchers for decades [104] [105], its original author is difficult to establish, as the technique is seldom mentioned in the literature. This is probably due to the technique's simplicity, which makes it appear as a mere programming trick. But the technique is of special interest here because it offers such a simple way of achieving great time-savings in meta-optimization, and gives an additional reason for using the weighted sum

approach to multi-objective optimization. Furthermore, the technique presented here is extended with sorting and is therefore less trivial than usual.

3.4.1 Compatible Optimization Methods

Greedy optimization methods are generally compatible with preemptive fitness evaluation, because they only move their optimizing agents in the case of strict improvement. Take for example the LUS method from chapter 2, which works by choosing a random point in the vicinity of its current position and moving to the new position only in the case of improvement to the fitness. Therefore the fitness evaluation of the new position can be aborted, as soon as it becomes known that it is actually worse than that of the current position, provided that it will not improve later during computation of the fitness.

Non-Greedy Optimization Methods

Some non-greedy optimization methods also provide support for preemptive fitness evaluation. For example the PSO method from chapter 2 also supports the technique, as the fitness is used greedily in determining whether or not to update the agent's best-known position, $\vec{p}$.

Although it requires more book-keeping, it is also possible to employ preemptive fitness evaluation to optimization methods such as HC and SA, whose movement decisions are stochastic according to the improvement in fitness. But those methods are of no particular concern in the remainder of the thesis and their compatibility with preemptive fitness evaluation is not detailed further.

3.4.2 Compatible Fitness Functions

Preemptive fitness evaluation is directly applicable to fitness functions (or meta-fitness functions) that are iteratively computed, and where the overhead of monitoring the progress and consequently aborting the fitness evaluation does not cancel out the gain in computation time that arises from only evaluating a part of the fitness function.

It is, however, a requirement for preemptive fitness evaluation to work that the fitness is accumulated in a non-decreasing manner because the fitness must only be allowed to grow worse during its computation, in order for the fitness computation to be aborted safely.

3.4.3 Application To Meta-Optimization

To employ preemptive fitness evaluation in meta-optimization, consider the algorithm for the meta-fitness measure given in figure 3.2. There, the results of optimizing a number of problems f_i are summed for the purpose of simultaneously evaluating the performance of an optimization method and a given choice of behavioural parameters on a broader set of optimization problems.

Ensuring Meta-Fitness Only Grows Worse

To ensure the meta-fitness measure is non-decreasing and can hence only grow worse, first assume the fitness functions f_i each have a known lower boundary denoted $\min(f_i)$. Since fitness results from several optimization runs are being added to form the meta-fitness measure, we must compensate for the possibility that one or more of these may be negative. This is done by subtracting $\min(f_i)$ from the individual fitness-contributions, thus ensuring they are all non-negative and the overall meta-fitness sum is therefore non-decreasing. If/when the accumulating meta-fitness becomes worse than what is required for the overlaying meta-optimizer to accept it as an improvement, it may be aborted preemptively.

Algorithm

The algorithm from figure 3.2 for computing the meta-fitness measure is modified to support preemptive fitness evaluation as shown in figure 3.3. Here, the preemptive fitness limit is denoted F and is the limit beyond which the meta-fitness evaluation can be aborted. This limit is passed as an argument by the overlaying meta-optimizer and is the meta-fitness of the agent that is potentially being replaced. This is made more clear in figure 3.4 which depicts the use of LUS as the meta-optimizer. In an actual implementation this can be made almost transparent, so an optimizer only needs minor modifications for it to work as a meta-optimizer with preemptive fitness evaluation. Also note that the choice of weights is $w_i = 1$, see section 3.3.1.

Evaluation Order

Some terms of the meta-fitness sum may be more likely than others to cause the evaluated meta-fitness to become worse than the preemptive fitness limit. A simple way of exploiting this to save even more computation time is to sort the evaluation order of the individual parts of the meta-fitness function, according to their last contributions to the overall meta-fitness. This sorting of the evaluation order occurs towards the end of the algorithm in figure 3.3.

To exemplify this, consider the term resulting from optimizing the Rosenbrock function, which is a benchmark problem that may give very large fitness values. The Ackley problem, on the other hand, has comparatively small fitness values. By sorting the order in which these benchmark functions are optimized, the problems that are most likely to cause the meta-fitness evaluation to be aborted will be optimized first.

Sorting is generally applicable to preemptive fitness evaluation, although it might work best when the individual subproblems require approximately the same time to execute, as is the case with these benchmark functions.

Unknown Fitness Boundary

For all practical purposes, one should be able to define a fitness boundary $\min(f_i)$ for all the problems f_i and hence be able to use the meta-fitness al-

gorithm above. In the rare case where this is not possible because the fitness boundary is virtually unknown, one can still employ preemptive fitness evaluation. This is done by changing the algorithm so that it first guesses the boundary and then adjusts both the meta-fitness boundary and all the preemptive fitness limits whenever it becomes known that the actual boundary is in fact lower. The book-keeping is more involved though, and since it will not be used in this thesis a detailed algorithm has been omitted.

Time-Savings Experienced

Depending on the experimental settings, the time-saving in meta-optimization caused by the use of preemptive fitness evaluation ranges from about 50% to a very substantial 85%. These estimates were computed from the experiments later in this thesis by extrapolating the actual time usage with the measured amount of preemptive abortion of the meta-fitness evaluations.

3.5 Noisy Meta-Fitness

The algorithm for computing the meta-fitness measure may cause two different results for two executions with the same input behavioural parameters. This is due to the stochastic nature of the optimizations that are being conducted inside that algorithm.

In general, fitness functions that change stochastically over time like this are called noisy. The overlaying meta-optimizer must be able to cope with such noise, in order to accurately find good choices of behavioural parameters for an optimizer by searching its noisy meta-fitness landscape. The following is a survey of techniques for coping with noisy problems in general and a discussion of which technique is most suitable for meta-optimization. A more extensive survey of different categories of time-varying fitness functions and how to optimize them with black-box methods can be found in [106].

In [107] DE is compared to other multi-agent optimization methods on a number of benchmark problems with Gaussian noise added. The study indicates that the DE method generally performs poorly on noisy optimization problems and may require modifications to alleviate this. Variants of optimization methods have indeed been developed to handle noisy fitness functions, or even non-stationary (also called dynamic) fitness functions – meaning the optimum moves around in the search-space over time. It seems reasonable that an optimization method being able to handle a non-stationary problem may also be able to handle a noisy problem such as the one associated with meta-optimization. An example of a PSO variant designed to cope with changing fitness functions is due to Carlisle and Dozier [108]. There, the PSO essentially resets its agents' previous best known positions, $\vec{p}$, if their corresponding fitness values have changed after a designated period of time. This is reported to help tracking a non-stationary optimum although an extra cost is incurred by having to reevaluate the fitness at intervals.

3.5.1 Averaging

A simple approach to lessen the effect of noise in a fitness function, without modifying the optimization method itself, is to repeat the fitness evaluation a number of times and use the average fitness value to guide the optimization.

It is suggested in [109] to schedule the number of times the fitness function is evaluated for a GA optimizing a noisy problem. This is done in an effort to lessen the effect of noise but at the same time decrease computation expense by only repeating fitness evaluations when necessary. Scheduling shows performance improvement on a number of noisy benchmark functions and is implemented in a two-fold manner. First, fitness noise is considered more acceptable early in an optimization run when the agents are spread out in the search-space and are more likely to have significantly different fitness values, thus making the effect of noise less significant. But as the optimization progresses and the fitness of the agents become more similar, more fitness evaluations have to be performed and averaged so as to lessen the effect of noise and more accurately tell the agents apart with regard to their fitness. The second idea is that since the scheduling is done for a GA in that paper and a GA uses selection of fitter agents in its creation of new agents, the number of times the noisy fitness function is evaluated can generally be made higher for the more fit agents as they are the most likely to be chosen for reproduction anyway.

The belief of noise being less harmful early in an optimization run is also used for the SA method in [110]. SA was briefly described in chapter 2 and is based on local sampling of a single agent using a stochastic movement rule. So noisy fitness functions are again suggested to be evaluated and averaged more times later in the optimization run, as was suggested for a GA above. This claim is backed by a mathematical proof in which convergence for the SA method on noisy fitness functions is shown to require a gradual increase in the number of times the fitness function is evaluated, so as to lessen the influence of noise. The proof in [110] only holds for combinatorial, that is, discrete optimization problems.

Studies with a GA on noisy fitness functions are conducted in [111] and indicate multi-agent optimization methods generally have an advantage over single-agent methods in the presence of noise. The results also suggest that population-size be increased and the number of fitness repeats decreased, as this seems to improve the results further. The cause of this appears to be an implicit averaging of fitness evaluations, which occurs due to the multitude of agents that are cooperating in optimization. This is echoed in [112] where it is shown theoretically that using a special GA selection scheme known as *Boltzmann* selection, the effect of Gaussian noise diminishes with an increasing number of agents. Indeed, in the theoretical case of an infinite number of agents, fitness noise is claimed to have no effect at all.

Stochastic deviations are already being lessened by repeating optimizations and summing the results in the meta-fitness algorithm from figure 3.2, and the algorithm in figure 3.3 employs Preemptive Fitness Evaluation to lessen the time-usage, while ensuring evaluations are repeated until meta-fitness improve-

ment can be determined correctly. This technique is simple to implement, works well, and is almost transparently supported by many black-box optimization methods. It is therefore preferred.

3.6 Choice of Meta-Optimizer

Because of the time usage involved in performing meta-optimization, the meta-optimizer must be able to achieve good results using few iterations. Ideally something like $20 \cdot n$ iterations will be allowed in a meta-optimization run, where n is the number of behavioural parameters to be tuned. For DE/rand/1/bin with $n = 3$ behavioural parameters this would mean a meta-optimization run consists of 60 iterations, that is, 60 evaluations of the meta-fitness algorithm in figure 3.3. This may also allow for the tuning of an optimizer's behavioural parameters with regard to problems that require more computation time. But it is a considerable demand for the meta-optimizer to search the meta-fitness landscape in so few iterations.

Considering the DE and PSO methods from chapter 2 and their optimization progress as plotted in figures 2.7-2.10, it is clear that these two optimizers with a standard choice of behavioural parameters are unable to achieve satisfactory results using so few iterations. Even with the better choice of DE parameters found by exhaustive search above, its optimization progress as reported in figures 3.11-3.12 still does not appear to be rapid enough for DE to be used as meta-optimizer. It may be possible to find behavioural parameters for DE and PSO that make them perform well when they are to be used as the meta-optimizer, but that would be meta-meta-optimization and we would be getting ahead of ourselves.

Smit and Eiben [33] conduct meta-optimization experiments using the *Covariance Matrix Adaptation* ES (CMA-ES) as the meta-optimizer. The CMA-ES is an optimization method originally due to Hansen and Ostermeier [113] that has become popular because it achieves good performance on benchmark problems, see e.g. García et al. [114] for a comparison to other optimizers, and CMA-ES does seem to have rapid optimization progress on a number of benchmark problems, see e.g. [113] [115], but CMA-ES is complicated to describe and implement if simpler methods could be used as the meta-optimizer instead. A more recent optimization method that has been made specifically for short optimization runs is known as SNOBFIT and is due to Huyer and Neumaier [116], however, it is also complicated to describe and implement.

The simple PS and LUS methods from chapter 2 were specifically designed to quickly optimize simpler problems and were often able to achieve good results on complex problems as well, as demonstrated in figures 2.11-2.14. From the simplicity of the meta-fitness landscapes shown in figures 3.5-3.10 it would therefore seem that either PS or LUS would be a good choice as the meta-optimizer.

3.6.1 Comparison of Meta-Optimizers

Experiments will now be conducted with the PS and LUS methods to establish which of them works best as the meta-optimizer, if at all. The DE/rand/1/bin optimizer from before will be used again, in particular its behavioural parameters NP and F are being tuned while the parameter CR is held fixed at $CR = 0.9$. Recall that PS and LUS sometimes have irregular performance and may stagnate prematurely, so five meta-optimization runs will be conducted in each experiment to overcome this, with each run consisting of $20 \cdot n = 40$ iterations of the meta-optimizer because $n = 2$ behavioural parameters are being tuned.

Figures 3.13-3.16 show the meta-optimization progress. The plots show the meta-fitness landscapes from figures 3.5-3.10 superimposed with the optimization moves made by either PS or LUS as the meta-optimizer. Their contemplated moves are shown as crosses and are the samples that PS and LUS make that result in worse meta-fitness so a move to that position is not made. The first thing to note is that PS has a tendency to focus its sampling while LUS spreads the sampling more in the search-space. In fact, PS often samples the same position several times, which is a quirk of that optimization method in lower dimensional search-spaces due to its halving of the step-size and reversal of direction that causes it to move back and forth to the same position several times. As the dimensionality of the search-space increases it gets increasingly unlikely that PS will sample the same position. This deficiency of PS in low-dimensional search-spaces could perhaps be remedied, but it seems unnecessary since LUS does not have this deficiency and can therefore be used instead. Concerning the results achieved in these meta-optimization runs, both PS and LUS locate the valley of good performing behavioural parameters for the DE optimizer. Table 3.3 shows the four best sets of behavioural parameters found in each meta-optimization experiment, note that these are not necessarily the results of four separate meta-optimization runs. The meta-fitness achieved from using PS or LUS as meta-optimizer is similar to that found in the grid-search of figures 3.5-3.10, although the location of the behavioural parameters are not precisely centered in the apparently optimal regions from the meta-fitness landscapes depicted. In one case the PS and LUS meta-optimizers actually found better results than the grid-search, this was for the experiment using all benchmark problems and allowing 60,000 optimization iterations. The grid-search yielded a meta-fitness of 12556 for its best found behavioural parameters, while table 3.3 shows that PS and LUS found behavioural parameters having a meta-fitness around 9500, although it may well be due to stochastic noise.

Because only five meta-optimization runs are being conducted in each experiment one should generally be wary of concluding too much from these experiments. It does seem, however, that there is no clear winner as PS sometimes works best (see the results with regard to all benchmark problems and 6,000 optimization iterations being allowed) and sometimes LUS works best (see the results with regard to Sphere & Rosenbrock and 6,000 optimization iterations being allowed). Overall PS and LUS seem to be somewhat comparable in terms

of the quality of the behavioural parameters they can find.

Concerning the time usage it is a different matter. From table 3.2 LUS has a clear advantage over PS as it is faster in performing meta-optimization. This is because Preemptive Fitness Evaluation works better for some optimization methods than others. As noted above, PS has a tendency in low-dimensional search-spaces to focus its sampling and sometimes sample the same position repeatedly, which will likely mean the meta-fitness evaluations have to be made in their entirety and cannot be preemptively aborted, as the meta-fitness values are too similar to be distinguished early in their evaluation. Regarding the time usage of doing meta-optimization compared to that of doing exhaustive search, see table 3.1, it is clear that meta-optimization requires only a small fraction of the time for doing a more exhaustive search for behavioural parameters.

For these reasons LUS seems to be the preferred choice for a meta-optimizer as it quickly found the behavioural parameters that made DE perform well in a number of optimization scenarios. LUS will therefore be used in the meta-optimization experiments for the remainder of the thesis.

3.7 Summary

This chapter gave a simple way of finding good behavioural parameters for an optimization method by employing another overlaying optimizer. This was named Meta-Optimization and made possible because of the choice of meta-optimizer, the LUS method from chapter 2, which requires comparatively few iterations to find good solutions, and by using a technique called Preemptive Fitness Evaluation for lowering the time usage further still. This meta-optimization setup will be used in the following chapters to make detailed studies of optimizer variants, now that their behavioural parameters can be tuned efficiently.

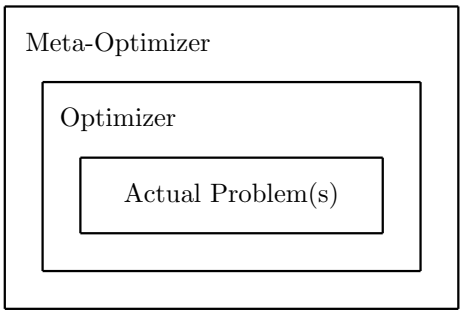

Figure 3.1: The concept of meta-optimization. A black-box optimizer is used in an offline manner as an overlaying meta-optimizer for finding good behavioural parameters of another optimization method, which in turn is used to optimize one or more actual problems.

- Initialize the problem-counter: $i \leftarrow 1$, and the fitness-sum: $s \leftarrow 0$.
- While $(i \leq N)$ iterate over the problems as follows:
  - Initialize the run-counter: $j \leftarrow 1$.
  - While $(j \leq M)$ do the following:
    - Perform an optimization run on problem f_i using the given optimization method with the given choice of behavioural parameters.
    - Add the best fitness obtained in the run, call it $f_i(\vec{g})$, to the fitness-sum: $s \leftarrow s + f_i(\vec{g}) \cdot w_i$, with w_i being a weight.
    - Increment the run-counter: $j \leftarrow j + 1$.
  - Increment the problem-counter: $i \leftarrow i + 1$.
- Return s as the meta-fitness value of the given optimization method and choice of parameters.

Figure 3.2: Meta-fitness algorithm for use in meta-optimization to rate the performance of a given optimization method and choice of behavioural parameters.

- Initialize the problem-counter: $i \leftarrow 1$, and the fitness-sum: $s \leftarrow 0$.
- While $(i \leq N)$ and $(s < F)$, do the following:
  - Initialize the run-counter: $j \leftarrow 1$.
  - While $(j \leq M)$ and $(s < F)$, do the following:
    - Perform an optimization run on problem f_i using the given optimization method with the given choice of behavioural parameters.
    - Add the best fitness obtained in the run, call it $f_i(\vec{g})$, to the fitness-sum, ensuring it is non-zero by subtracting $\min(f_i)$:

      $$s \leftarrow s + (f_i(\vec{g}) - \min(f_i)) \cdot w_i$$

      with w_i being a weight.
    - Increment the run-counter: $j \leftarrow j + 1$.
  - Increment the problem-counter: $i \leftarrow i + 1$.
- Sort the problems f_i descendingly according to their contributions to the overall fitness sum s. This will likely allow for earlier preemptive abortion of the next meta-fitness evaluation.
- Return s as the meta-fitness value of the given optimization method and choice of behavioural parameters.

Figure 3.3: Meta-fitness algorithm with preemptive evaluation used to rate the performance of a given optimization method and choice of behavioural parameters. This algorithm is the same as figure 3.2 above, but employing Preemptive Fitness Evaluation to save computation time.

- Initialize $\vec{x}$ to a random choice of behavioural parameters for the optimizer whose parameters are to be tuned. For DE/rand/1/bin this would be a random choice of $\vec{x} = (NP,\, CR,\, F)$
- Compute the meta-fitness of the choice of behavioural parameters $\vec{x}$ using the algorithm in figure 3.3, by passing the limit $F = \infty$ as an argument to ensure full evaluation of the meta-fitness. Call the meta-fitness so computed for $f(\vec{x})$ and store it for later use.
- Set the initial sampling range $\vec{d}$ to cover the entire space of behavioural parameters for the optimizer whose parameters are being tuned.
- Until a termination criterion is met, such as a number of iterations having been performed, or a threshold for $f(\vec{x})$ has been reached, repeat the following:
  - Pick a random vector $\vec{a} \sim U\left(-\vec{d}, \vec{d}\right)$
  - Add this to the behavioural parameters $\vec{x}$ to create the new potential choice of behavioural parameters, call these $\vec{y}$:

    $$\vec{y} = \vec{x} + \vec{a}$$

  - Compute the meta-fitness of the behavioural parameters $\vec{y}$ by using the algorithm in figure 3.3 and by setting the preemptive fitness limit $F = f(\vec{x})$, because the meta-fitness evaluation can be aborted if it is worse than that needed for the new parameters $\vec{y}$ to replace the currently best known parameters $\vec{x}$, see next step.
  - If $(f(\vec{y}) < f(\vec{x}))$ then update the best known choice of behavioural parameters:

    $$\vec{x} \leftarrow \vec{y}$$

    Otherwise decrease the sampling-range by multiplication with the factor q from Eq.(2.1):

    $$\vec{d} \leftarrow q \cdot \vec{d}$$

Figure 3.4: Using the LUS method as meta-optimizer.

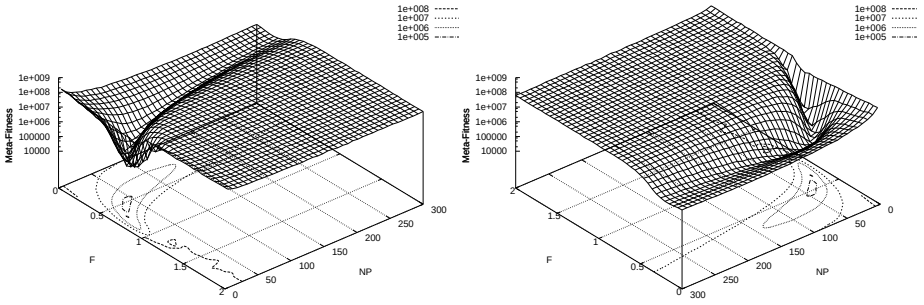

Figure 3.5: Meta-fitness landscape shown at different angles for DE/rand/1/bin computed by varying the NP and F parameters and keeping a fixed parameter $CR = 0.9$, measuring the performance of DE on **Sphere and Rosenbrock** in 30 dimensions over 50 optimization runs of **6,000** fitness evaluations each.

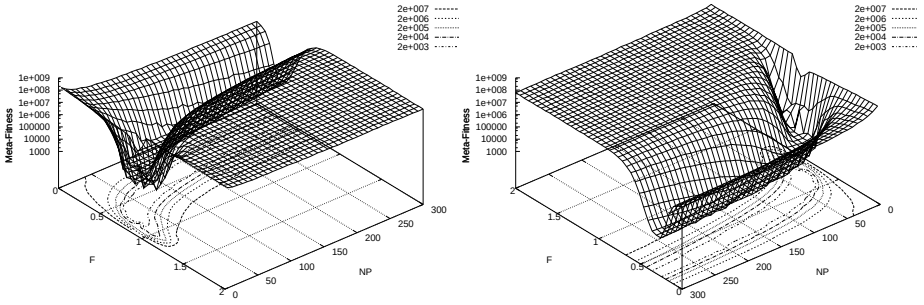

Figure 3.6: Meta-fitness landscape shown at different angles for DE/rand/1/bin computed by varying the NP and F parameters and keeping a fixed parameter $CR = 0.9$, measuring the performance of DE on **Sphere and Rosenbrock** in 30 dimensions over 50 optimization runs of **60,000** fitness evaluations each.

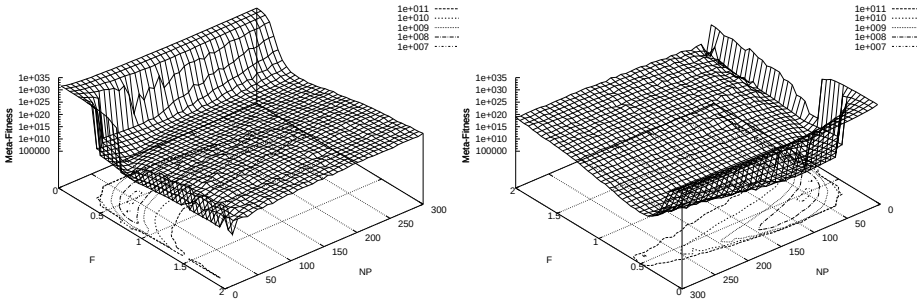

Figure 3.7: Meta-fitness landscape shown at different angles for DE/rand/1/bin computed by varying the NP and F parameters and keeping a fixed parameter $CR = 0.9$, measuring the performance of DE on all **12 benchmark problems** in 30 dimensions over 50 optimization runs of **6,000** fitness evaluations each.

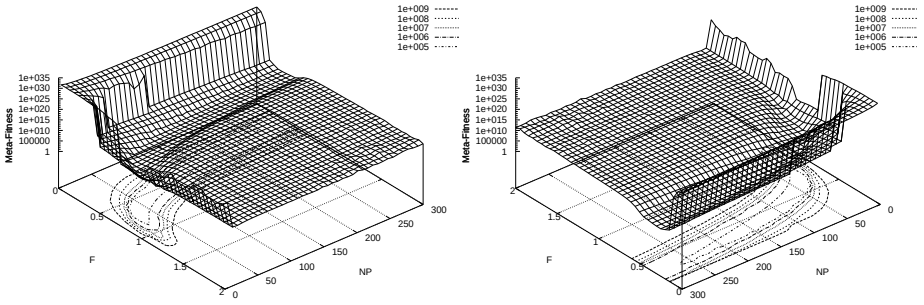

Figure 3.8: Meta-fitness landscape shown at different angles for DE/rand/1/bin computed by varying the NP and F parameters and keeping a fixed parameter $CR = 0.9$, measuring the performance of DE on all **12 benchmark problems** in 30 dimensions over 50 optimization runs of **60,000** fitness evaluations each.

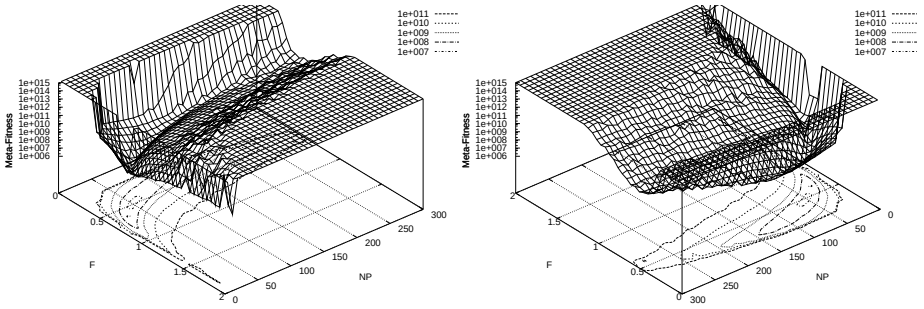

Figure 3.9: Meta-fitness landscape shown at different angles for DE/rand/1/bin computed by varying the NP and F parameters and keeping a fixed parameter $CR = 0.9$, measuring the performance of DE on all **12 benchmark problems** in 30 dimensions over 50 optimization runs of **6,000** fitness evaluations each. Meta-fitness has been **capped** at 1e+15 for clarity.

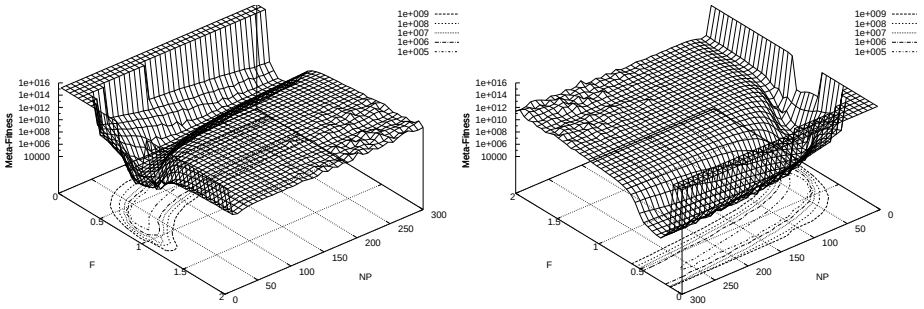

Figure 3.10: Meta-fitness landscape shown at different angles for DE/rand/1/bin computed by varying the NP and F parameters and keeping a fixed parameter $CR = 0.9$, measuring the performance of DE on all **12 benchmark problems** in 30 dimensions over 50 optimization runs of **60,000** fitness evaluations each. Meta-fitness has been **capped** at 1e+15 for clarity.

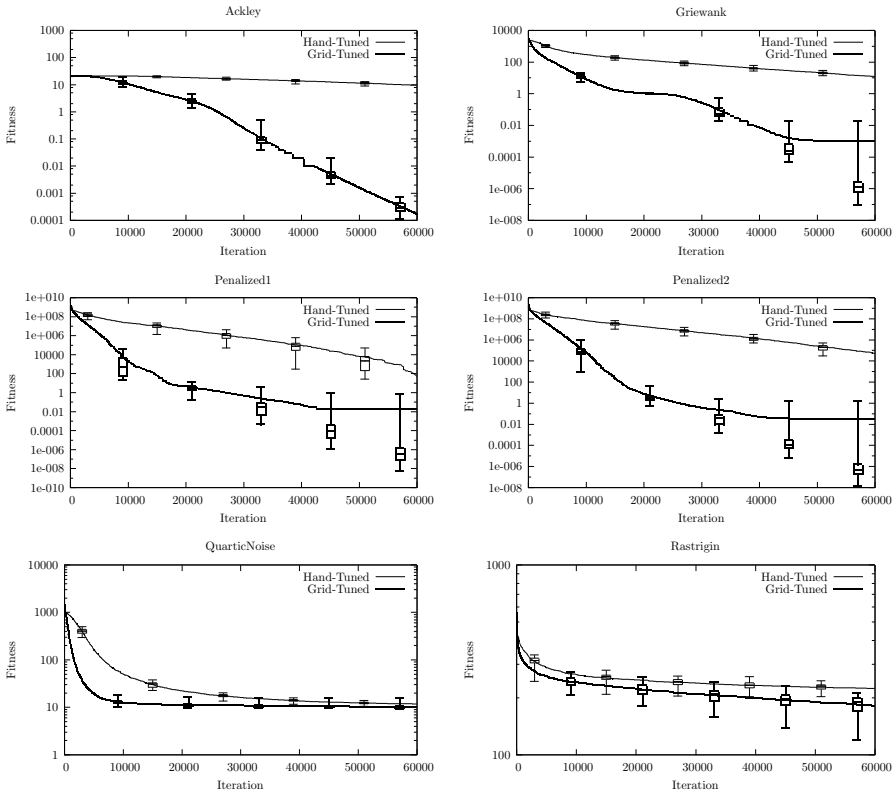

Figure 3.11: Comparison of optimization progress for **DE/rand/1/bin** using the **hand-tuned** behavioural parameters $NP = 300$, $CR = 0.9$, $F = 0.5$ and the **grid-tuned** parameters $NP = 50$, $CR = 0.9$, $F = 0.6$. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

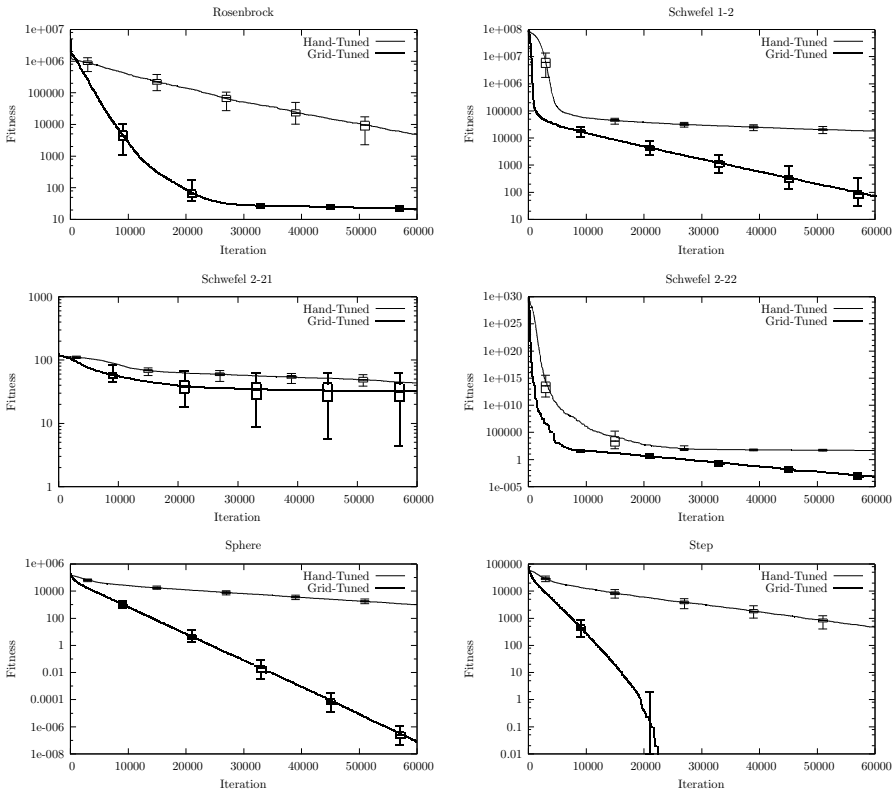

Figure 3.12: Comparison of optimization progress for **DE/rand/1/bin** using the **hand-tuned** behavioural parameters $NP = 300$, $CR = 0.9$, $F = 0.5$ and the **grid-tuned** parameters $NP = 50$, $CR = 0.9$, $F = 0.6$. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization. Note that a fitness value of zero is reached for the Step problem, which cannot be plotted on a logarithmic scale.

Configuration	6,000 Iterations	60,000 Iterations
Sphere & Rosenbrock	1 hours 22 min	14 hours 04 min
All Benchmark Problems	12 hours 01 min	119 hours 18 min

Table 3.1: Time usage for computing a 40x40 grid of meta-fitness values for the DE/rand/1/bin optimizer by varying the behavioural parameters NP and F while keeping a fixed parameter $CR = 0.9$. Plots of the resulting grids can be seen as meta-fitness landscapes in figures 3.5-3.10.

Problems	Optimization Iterations	Meta-Optimizer	Time Usage
Sphere & Rosenbrock	6,000	PS	9 min
		LUS	5 min
	60,000	PS	1 h 01 min
		LUS	35 min
All Benchmark Problems	6,000	PS	26 min
		LUS	22 min
	60,000	PS	5 h 36 min
		LUS	3 h 07 min

Table 3.2: Time usage for meta-optimizing DE/rand/1/bin with either the PS or LUS as meta-optimizer and with different configurations of benchmark problems and optimization run-lengths. The DE parameters NP and F are being tuned while keeping a fixed parameter $CR = 0.9$. Plots of these meta-optimization runs can be seen in figures 3.13-3.16.

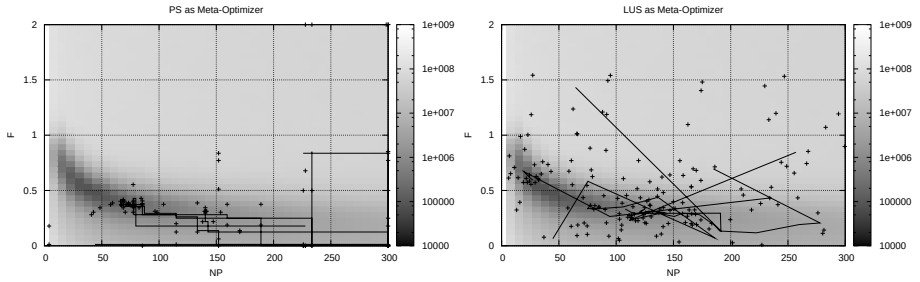

Figure 3.13: Meta-optimization progress of PS and LUS finding good behavioural parameters NP and F of DE/rand/1/bin with a fixed parameter $CR = 0.9$, when DE is used for optimizing the **Sphere and Rosenbrock** problems in 30 dimensions over 50 optimization runs of **6,000** fitness evaluations each. Lines show moves made by the meta-optimizer, crosses show moves contemplated but not taken as they would lead to worse meta-fitness. These are superimposed on the meta-fitness landscape from figure 3.5 to show the valley of good performing parameters is found.

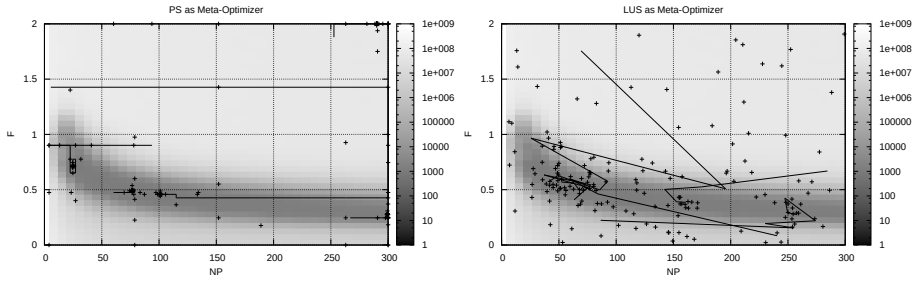

Figure 3.14: Meta-optimization progress of PS and LUS finding good behavioural parameters NP and F of DE/rand/1/bin with a fixed parameter $CR = 0.9$, when DE is used for optimizing the **Sphere and Rosenbrock** problems in 30 dimensions over 50 optimization runs of **60,000** fitness evaluations each. Lines show moves made by the meta-optimizer, crosses show moves contemplated but not taken as they would lead to worse meta-fitness. These are superimposed on the meta-fitness landscape from figure 3.6 to show the valley of good performing parameters is found.

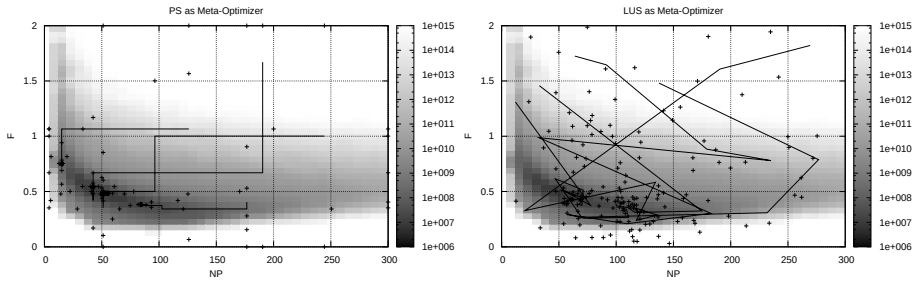

Figure 3.15: Meta-optimization progress of PS and LUS finding good behavioural parameters NP and F of DE/rand/1/bin with a fixed parameter $CR = 0.9$, when DE is used for optimizing all **12 benchmark problems** problems in 30 dimensions over 50 optimization runs of **6,000** fitness evaluations each. Lines show moves made by the meta-optimizer, crosses show moves contemplated but not taken as they would lead to worse meta-fitness. These are superimposed on the meta-fitness landscape from figure 3.7 to show the valley of good performing parameters is found. The meta-fitness landscape has been capped at 1e+15 for clarity.

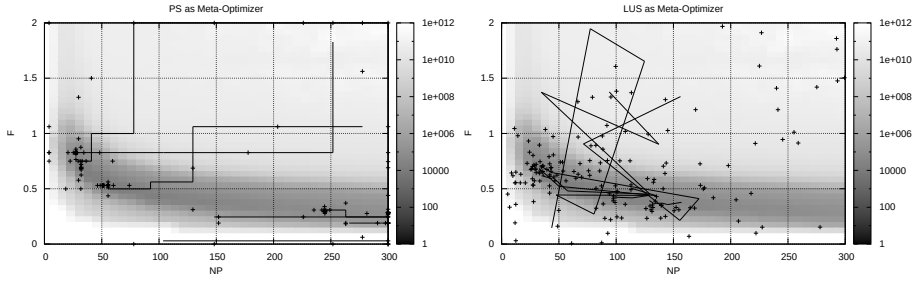

Figure 3.16: Meta-optimization progress of PS and LUS finding good behavioural parameters NP and F of DE/rand/1/bin with a fixed parameter $CR = 0.9$, when DE is used for optimizing all **12 benchmark problems** problems in 30 dimensions over 50 optimization runs of **60,000** fitness evaluations each. Lines show moves made by the meta-optimizer, crosses show moves contemplated but not taken as they would lead to worse meta-fitness. These are superimposed on the meta-fitness landscape from figure 3.8 to show the valley of good performing parameters is found. The meta-fitness landscape has been capped at 1e+12 for clarity.

Problem Configuration			Parameters Found		
Problems	Optimization Iterations	Meta-Optimizer	NP	F	Meta-Fitness
Sphere & Rosenbrock	6,000	PS	69	0.4029	124565
			69	0.3951	131087
			69	0.3873	134304
			73	0.3750	150770
		LUS	27	0.5940	61689
			25	0.6137	65659
			19	0.6714	86009
			78	0.3808	175844
	60,000	PS	25	0.7138	1063
			77	0.4909	1142
			77	0.5066	1147
			79	0.5066	1172
		LUS	52	0.5693	1069
			60	0.5621	1115
			42	0.5763	1115
			61	0.5542	1124
All Benchmark Problems	6,000	PS	52	0.4819	4.53e+6
			42	0.5455	4.56e+6
			52	0.4858	4.67e+6
			55	0.4857	4.75e+6
		LUS	51	0.5154	6.75e+6
			64	0.4637	7.02e+6
			64	0.4701	8.41e+6
			72	0.4361	8.81e+6
	60,000	PS	32	0.6875	9518
			51	0.5304	13168
			53	0.5304	13494
			56	0.5304	13582
		LUS	35	0.6193	9529
			43	0.5928	11682
			37	0.6113	11968
			33	0.7104	12716

Table 3.3: Results of meta-optimizing behavioural parameters NP and F of DE/rand/1/bin with a fixed parameter $CR = 0.9$, for a variety of scenarios: Different selections of benchmark problems, different number of optimization iterations (6,000 and 60,000), and PS and LUS methods used as meta-optimizers.The progress of the meta-optimization runs leading up to these results are shown in figures 3.13-3.16.

Chapter 4

Differential Evolution

4.1 Introduction

This chapter studies state-of-the-art variants of the DE optimization method and how they perform using different choices of behavioural parameters in various optimization scenarios. A simplified DE variant is also studied to see if performance is impaired or improved by simplifying the DE algorithm.

4.1.1 Background

Much research effort has been put into improving the performance of DE. The behavioural parameters of DE were analyzed mathematically by Zaharie [30] who aimed at keeping the optimizing agents from converging prematurely by preserving their diversity in the search-space. The analysis yielded conditions for the behavioural parameters to meet, although due to the limiting assumptions of the analysis the empirical results were not a complete success.

Attempts have also been made at eliminating the need for tuning the behavioural parameters of DE altogether by perturbing or adapting the parameters during optimization. It seems there are two reasons for doing this. First is the belief that it remedies the need for a user to manually select parameters that yield good performance on a problem at hand. Second is the belief that different choices of parameters are needed at different stages of optimization so as to balance exploration and exploitation of the search-space. Otherwise the parameters could just as well be held fixed during optimization. Both these beliefs will be studied in this chapter.

One technique for doing such perturbing of the behavioural parameters of DE is suggested by Kaelo and Ali [117]. It consists of picking the behavioural parameters at random in each iteration, from the parameter ranges which have been found in the literature to work well. They furthermore propose the use of the *Electromagnetism-Like* (EM) method by Birbil and Fang [118], so as to form a hybrid DE and EM method, where the attraction-repulsion feature of

EM is combined with DE. Although this new DE hybrid is reported to improve performance on a number of benchmark problems, it is not only significantly more complicated than the basic DE method, but also fails to eliminate the behavioural parameters as it actually introduces new parameters in form of the boundaries for the stochastic sampling ranges of the DE parameters. Furthermore, the convergence proof of the EM method by Birbil et al. [21] is a limit-proof, meaning that convergence to the global optimum occurs with probability one, provided optimization is allowed to run for sufficiently many iterations. But such a proof is useless in practice because the exact same thing can be shown for completely random sampling as mentioned in chapter 1.

The Fuzzy Adaptive DE (FADE) by Liu and Lampinen [36] uses Fuzzy Logic to adapt the DE behavioural parameters during an optimization run. Fuzzy logic, originally due to Zadeh [37], provides a means for logical reasoning with uncertainties. In FADE the fuzzy reasoning is used to alter DE behavioural parameters according to observations of fitness improvements and population diversity and is reported to outperform DE with fixed and hand-tuned behavioural parameters, especially on benchmark problems with higher dimensionalities.

Another example of an adaptive DE variant is the Self-adaptive DE (SaDE) due to Qin and Suganthan [38]. The SaDE variant perturbs the F parameter according to a normal distribution, and while the CR parameter is also randomly picked, its observed effect on fitness improvement influences how and when this random picking occurs. Additionally, SaDE uses several DE variants (e.g. DE/rand/1/bin and DE/current-to-best/1/bin, see [119]) which are switched between in a stochastic manner according to their observed ability to improve the fitness during optimization. Note that this is significantly more complicated than the original DE method and will require a good deal more effort in implementation, which must be made up for by improved performance.

4.2 Dither & Jitter Variants

Several schemes for perturbing the differential weight F have been proposed in the literature, see for example [25] [119]. The ones used by Storn himself are generally the simple Dither and Jitter schemes [78] [120] which will also be used in this study. In the Dither scheme a random differential weight F is picked on a per-agent basis:

$$F = F_l + r' \cdot (F_u - F_l) \tag{4.1}$$

so $r' \sim U(0,1)$ is a random number picked once for each agent being updated using Eq.(2.2). Another way of perturbing the differential weight is to use the Jitter scheme, also from [25] [120]:

$$F_i = F_{mid} \cdot (1 + \delta \cdot (r'_i - 0.5))$$

where $r'_i \sim U(0,1)$ is now being picked for each element of the vector being updated in Eq.(2.2). The parameter δ determines the scale of perturbation and will be eliminated shortly. For although these formulae appear to be distinct

at first glance they are in fact equivalent, with the exception of Dither drawing a random differential weight once for each agent-vector to be updated, and Jitter drawing a random weight for each element of that vector. To see this let $F_l = F_{mid} \cdot (1 - \delta/2)$ and $F_u = F_{mid} \cdot (1 + \delta/2)$ in Eq.(4.1). A simpler and more mathematical description of Dither would therefore be:

$$F \sim U(F_l, F_u)$$

and for Jitter:

$$F_i \sim U(F_{mid} \cdot (1 - \delta/2), F_{mid} \cdot (1 + \delta/2))$$

Using a midpoint and a range instead of perturbation boundaries has the advantage of being independent from each other, whereas the F_l and F_u boundaries must also satisfy $F_l < F_u$, which would make automatic tuning of these more difficult. A common notation will therefore be used here for both Dither and Jitter, having a midpoint F_{mid} and a range F_{range}. For Dither this means the differential weight is picked randomly as:

$$F \sim U(F_{mid} - F_{range},\, F_{mid} + F_{range})$$

and for Jitter this would be:

$$F_i \sim U(F_{mid} - F_{range},\, F_{mid} + F_{range})$$

meaning that F_i should be drawn once for every vector-element i in Eq.(2.2), as opposed to once for the entire vector when using the Dither scheme.

The intention of perturbing the differential weight F was to free the user from selecting a good value for F. But it has actually introduced two new behavioural parameters which must be selected by the user, F_{mid} and F_{range} determining the limits of perturbation for F. The midpoint can be anywhere in $F_{mid} \in [0, 2]$, and the perturbation range can be anywhere in $F_{range} \in [0, 3]$, which were chosen that wide to allow for unusual values when F_{mid} and F_{range} will be automatically tuned later in this study. Note, however, that this also allows for negative differential weights to occur, which for Jitter means the computation of some vector-elements in Eq.(2.2) might use a positive differential weight and for others a negative weight. The semantic meaning of this and how it influences optimization performance is impossible to foresee.

Concerning stochastic behavioural parameters it seems to be the belief that replacing fixed behavioural parameters with stochastic ones makes it easier for a user to tune the optimizer's behaviour, because the choosing of boundaries for the stochastic parameters affect optimization performance more leniently than choosing the actual parameters. Perhaps the belief is that sooner or later a good choice of behavioural parameter will be chosen at random. But one has to remember that completely random sampling of anything, whether it be behavioural parameters or candidate solutions to the optimization problem at hand, statistically takes a great many samples before finding a good choice. Therefore one might question if perturbation of parameters such as done in the Dither and Jitter schemes has a real chance of finding good choices of

behavioural parameters, or whether such perturbation just slightly alters the behaviour of the DE agents, without having a generally adaptive effect on the behavioural parameters that would make DE perform well on mostly any given optimization problem. This will be studied later.

4.3 JDE Variant

In addition to the perturbing DE variants above the adaptive DE variant known as JDE will also be used, which is due to Brest et al. [39]. This variant has been chosen because its performance has been found to compare well against other allegedly state-of-the-art, adaptive DE variants [39] [121]. It is presented by its authors as a Self-Adaptive DE as well because it eliminates the need for a user to select the F and CR parameters; yet ironically introduces 8 new user-adjustable parameters to achieve this. But this tendency is common for adaptive DE variants and indeed also for the comparatively simpler Dither and Jitter variants as described above, which merely perturb a behavioural parameter.

The JDE variant works as follows. First assign start values to F and CR, call these F_{init} and CR_{init}, respectively. Then before computing the new potential position of a DE agent using Eq.(2.2), first decide what parameters F and CR to use in that formula. With probability $\tau_F \in [0, 1]$ draw a new random $F \sim U(F_l, F_l + F_u)$, otherwise reuse F from previously, where each DE agent retains its own F parameter. Similarly, each agent retains its own CR parameter for which a new random value $CR \sim U(CR_l, CR_l + CR_u)$ is picked with probability $\tau_{CR} \in [0, 1]$ and otherwise the old CR value for that agent is reused. Whichever F and CR values are being used in the computation of Eq.(2.2), they will survive to the next iteration or be discarded along with the agent's new potential position $\vec{y}$ according to fitness improvement.

In the original description of JDE [39] the authors encode each agent's F and CR values in that agent's candidate solution vector $\vec{x}$ by extending the vector accordingly. But since these F and CR values never themselves undergo the crossover and mutation computation in Eq.(2.2), this encoding is not only unnecessary but it makes the presentation hard to comprehend. It would have been difficult if not impossible to implement JDE from the published description in [39] had its authors not supplied the actual source-code. This is a good example of an optimization method which is perhaps made more complex than it ought to be, because heuristical optimizers cannot currently be proven correct by analytical means.

4.4 Experimental Results

4.4.1 Standard Behavioural Parameters

The purpose of this experiment is to establish how well these DE variants fare against each other with standard choices of behavioural parameters. The parameters are shown in table 4.2 and are standard in the literature [16] [17] [119] [39]. They come from a variety of both manual and automated experiments with choosing behavioural parameters. The experience from section 3.3.2 has been used in choosing two of the three parameters for the basic DE/rand/1/bin variant, namely NP and F, as well as setting the population size $NP = 50$ instead of 300 for the Dither and Jitter variants, which is otherwise common in the literature. What is meant by standard DE parameters is therefore the currently best known behavioural parameters including the experience from chapter 3.

Concerning the end results achieved, as shown in table 4.4, the first thing to note is that they are fairly similar overall, with a DE variant performing a little better than the other DE variants on some problems and a little worse on others. For example, the Dither and JDE variants perform poorly on the Schwefel1-2 problem which Jitter is best at optimizing. But JDE performs better on Rastrigin. Concerning the average performance progress as shown in figures 4.1 and 4.2 the same holds true, for instance, the Jitter variant has best progress on the Rosenbrock problem and worst progress on the Schwefel2-21 problem. Another interesting thing to note is that the average optimization progress seems to be fairly similar in curvature on some of these problems, see for instance the QuarticNoise, Schwefel1-2, and Schwefel2-22 problems, this suggests that the inner workings of these DE variants are perhaps somewhat similar in spite of their algorithm differences.

4.4.2 Overall Meta-Optimized Performance

The purpose of this experiment is to establish how well the DE variants perform when their behavioural parameters have been tuned for all 12 benchmark problems simultaneously, with the DE variants being allowed 60,000 iterations for each optimization run on each of the problems. The boundaries for the behavioural parameters are shown in table 4.1 where the population size NP is now bounded to 200 agents following the experience in chapter 3 where good choices of NP were found to be below that. The parameters thus found through meta-optimization are shown in table 4.3. Comparing these to the standard parameters in table 4.2 shows several differences. For the basic DE variant the population size is now $NP = 19$ which is considerably lower than the 300 agents originally suggested in the literature and also lower than the 50 agents found to be a good choice in chapter 3. Here the parameter CR was also being meta-optimized whereas in chapter 3 it was held fixed at $CR = 0.9$ as recommended in the literature. The meta-optimized parameters have $CR = 0.122$ which apparently gives a different influence of the population size NP on optimization performance, hence the smaller $NP = 19$. For the Dither and Jitter variants the CR

and F_{mid} parameters are fairly similar to those recommended in literature, but their ranges of perturbation, F_{range}, are significantly greater than recommended in literature, especially for the Jitter variant where $F_{range} = 0.0005$ was recommended but meta-optimization found a good choice being $F_{range} = 0.3426$. As for the parameters of JDE they are more difficult to interpret because their intrinsic meaning is opaque. However, it can be seen easily that the meta-optimized JDE parameters are significantly different from the standard choice recommended in the literature.

Using the DE variants with the meta-optimized parameters results in table 4.5 which will now be compared to the results in table 4.4 from using standard behavioural parameters. Recall that the performance measure that was being tuned for in meta-optimization was the sum of the fitness achieved in 50 optimization runs on each of these benchmark problems using 60,000 iterations. This means the measures in table 4.5 that ought to benefit most from meta-optimization are the mean fitness values in an overall manner for all the benchmark problems. Negligible differences in performance will be ignored in the following commentary because these are only stochastic benchmark results which should not be taken as precise measurements of performance.

When using meta-optimized parameters the basic DE variant has improved the mean fitness dramatically on the Rastrigin problem but it has worsened just as dramatically on the Schwefel1-2 problem. The other results are sometimes a little better and sometimes a little worse when compared to the results of using standard parameters. For the Dither variant the mean fitness results using meta-optimized parameters are mostly an improvement, especially on the Rastrigin and Schwefel1-2 problems. The results for the Jitter variant are not as clear since the mean fitness achieved on the Penalized2 and Rastrigin problems is improved while it is greatly worsened on the Penalized1 and Schwefel1-2 problems. However, the quartiles on these problems are comparable to before so the performance difference is due to irregularity in the results of only a few optimization runs. For the JDE variant the mean fitness results are overall improved using the meta-optimized parameters, especially for the Rastrigin and Schwefel1-2 problems. It is true that JDE has worse mean fitness on some problems, e.g. the Ackley and Griewank problems, but the differences are negligible considering the quartiles and the fact that they have optimal fitness values of zero and the results are close to that.

The first conclusion to be made is that the meta-optimized behavioural parameters seem to perform roughly on par with the standard parameters, sometimes improving the results slightly and sometimes worsening them. This would suggest that the standard parameters were well suited for this test bed of benchmark problems with a dimensionality of 30 and optimization run-lengths of 60,000 iterations. This does not rule out meta-optimization as a useful technique for finding behavioural parameters, rather the opposite because no human effort is needed for doing meta-optimization whilst much human effort has been put into finding the guidelines for choosing DE parameters by hand and human effort will be needed in continually applying these guidelines. Additionally, the parameters here termed standard were in fact using a choice of population size

$NP = 50$ which was found in chapter 3 using exhaustive search of a grid of parameter combinations to be much better than $NP = 300$ as recommended in literature. So the standard parameters used here are not entirely hand-tuned but have been significantly refined by automated tuning, which must be taken into account when comparing the effort that has been put into finding the standard versus the meta-optimized parameters.

The other conclusion to be made regards the performance of the DE variants compared to each other. From the end results in table 4.5 it now appears that JDE is the best performing DE variant. Although in the progress plots in figures 4.3 and 4.4 it seems that JDE achieves its better results half-ways or later towards the end of the optimization runs and does not have any advantage earlier during optimization where it performs roughly on par with the other DE variants, except perhaps for the basic DE variant which is better early during optimization on some of these problems. It is also important to stress that JDE only became the top performing DE variant with regard to the end results in table 4.5, after it had its behavioural parameters meta-optimized. This is ironic because JDE was specifically designed so as not to need such parameter tuning as the JDE parameters were intended to adapt during optimization. Figures 4.5 and 4.6 show in more detail how meta-optimization has mostly improved JDE performance, the exception being the mean fitness progress on the Griewank and Schwefel2-21 problems, which, however, can be seen from the quartiles to be irregular and the tuned parameters do often but not always achieve significantly better fitness values than using the standard parameters.

The remainder of this chapter will use JDE in performance comparisons because it was found here to be the one achieving best end results overall and because its complicated algorithm is in stark contrast to the simplified DE that will be introduced next.

4.4.3 Simplification

This section introduces a simplification to the DE optimization method. The DE/*rand*/1/bin family of optimizers has been used in the previous experiments due to its popularity. The DE/*best*/1/bin variant on the other hand, has been long out of favour with researchers and practitioners because it is believed to have inferior performance with tendencies for premature convergence [25] [78]. The DE/best/1/bin replaces Eq.(2.2) with the following:

$$y_i = \begin{cases} g_i + F \cdot (a_i - b_i) & , (i = R) \text{ or } (r_i < CR) \\ x_i & , \text{else} \end{cases} \tag{4.2}$$

where $\vec{g}$ is the population's best known position in the search-space until now. In the original version of this, the agents $\vec{a}$ and $\vec{b}$ are chosen to be different not only from each other, but also from the agent $\vec{x}$ currently being processed. It simplifies the implementation a good deal if this is not required and only $\vec{a}$ and $\vec{b}$ must be different. In particular, first the index r_a for agent $\vec{a}$ is picked randomly from $\{1, \cdots, NP\}$, and then the index r_b for the other agent is determined by

first picking a random $r'_b \in \{1, \cdots, NP-1\}$ and then computing:

$$r_b = \begin{cases} r_a + r'_b & , r_a + r'_b \leq NP \\ r_a + r'_b - NP & , \text{else} \end{cases}$$

Furthermore, instead of iterating over every agent in the population as done in the original DE algorithm in figure 2.3, it is easier to also randomly pick the agent $\vec{x}$ to be updated. The DE variant with these simplifications will be called DE/simple.

Meta-optimization will be used to find good choices of behavioural parameters for DE/simple and the search-space for the behavioural parameters is bounded same as for DE/rand/1/bin, as shown in table 4.1. Meta-optimizing the behavioural parameters of DE/simple so as to perform well on all 12 benchmark problems when allowed 60,000 optimization iterations, yields:

$$NP = 186,\ CR = 0.8493,\ F = 0.4818 \tag{4.3}$$

Comparing these parameters to the ones tuned for basic DE/rand/1/bin as shown in table 4.3, there is a strikingly big change in population size from $NP = 19$ to $NP = 186$ and the crossover probability CR being almost inverted from having been $CR = 0.1220$ for the basic DE variant to now being $CR = 0.8493$ for the simplified DE. The differential weight F is similar to before, though. This is interesting because it shows that fairly small changes and simplifications made to an optimization algorithm can cause rather significant changes to the behavioural parameters that makes it perform well.

Table 4.6 shows the end results of using DE/simple and the JDE variant. It is first noted that the simplified DE cannot optimize Ackley at all within the number of iterations allowed and it also does not perform too well on Rastrigin. But on the Rosenbrock, Schwefel1-2 and Schwefel2-21 problems the simplified DE achieves significantly better results than JDE as it often comes close to the optimal fitness value of zero and particularly the Rosenbrock and Schwefel1-2 problems are considered hard problems. It is true that DE/simple has slightly worse average performance on e.g. Penalized1 and 2, but the quartiles reveal that it often finds solutions to these problems of better fitness than those found by JDE. As noted previously, however, these are stochastic benchmark results and should not be considered as precise measurements of performance capabilities, so when an optimizer finds solutions close to the optimal fitness value and there is some small irregularity it is hard to conclude that one optimizer is better than the other. The performance of DE/simple and JDE on e.g. the Penalized1 and 2 problems therefore seems to be comparable.

Concerning the optimization progress of DE/simple versus JDE as shown in figures 4.7 and 4.8, it is obvious that DE/simple stagnates quickly on the Ackley and Rastrigin problems. On the other hand it can also be seen that DE/simple progresses better than JDE earlier during optimization on most of the benchmark problems. Concerning regularity as depicted by the quartiles in these plots, the optimization progress might seem quite irregular at a first glance, but it should be kept in mind that the fitness axes are logarithm scaled

so what might appear as a big irregularity in e.g. the Griewank plot is actually solutions that are all fairly close the optimal fitness value.

The experiments have shown that the DE/simple variant, which keeps its behavioural parameters fixed during optimization, can have its parameters tuned so as to perform on par with the more sophisticated JDE/rand/1/bin variant, which attempts to adapt its behavioural parameters during optimization. On some of the benchmark problems DE/simple holds the advantage and on other problems JDE is best.

4.4.4 Short Optimization Runs

The purpose of this experiment is to establish how the DE/simple and JDE optimizer variants perform when their behavioural parameters have been tuned for 6,000 optimization iterations instead of 60,000 as before. Tuning the behavioural parameters with regard to all 12 benchmark problems results in the parameters shown in table 4.7. Comparing the DE/simple parameters to those found in Eq.(4.3) when allowing 60,000 optimization iterations shows that a somewhat smaller population size is now being used, $NP = 136$ as opposed to $NP = 186$, a crossover probability CR of almost one as opposed to $CR \simeq 0.85$, and a lower differential weight $F \simeq 0.28$ as opposed to $F \simeq 0.48$. It is unknown why the parameters should be chosen differently like this and the only noteworthy thing is perhaps that the population size still remains rather high at $NP = 136$ even though only 6,000 optimization iterations are being allowed. Comparing the JDE parameters to the ones tuned for 60,000 optimization iterations as shown in table 4.3, the only parameter that is roughly the same is the population size NP whilst the other parameters are very different. The intrinsic meaning of this parameter change is unknown as the inner workings of the JDE algorithm are even more opaque than those of DE/simple.

Representative optimization results of using these behavioural parameters are shown in figures 4.9 and 4.10. What is interesting in these plots is what happens to the optimization progress both before, at, and after the point of 6,000 optimization iterations for which the performance was sought tuned.

For DE/simple as shown in figure 4.9 the performance is improved overall on the Schwefel1-2 problem when the behavioural parameters are tuned for 6,000 iterations instead of 60,000. That is, the performance is improved both before and after this number of optimization iterations. On the Schwefel2-21 problem the inverse is true because the performance is now worse both before and after the 6,000 iterations mark. On the Rastrigin problem the performance is improved early during optimization at around 2000 iterations, but worsened for the remainder of the optimization run, including a worse performance at the 6,000 iterations mark for which the behavioural parameters were tuned. On the Schwefel2-22 problem there is an improvement before and somewhat after the 6,000 iterations mark but stagnation then occurs and the behavioural parameters tuned for 60,000 iterations gradually start to perform better and finally yield significantly better fitness values. These results are representative for all the benchmark problems when comparing the performance of behavioural pa-

rameters for DE/simple that were tuned for either 60,000 or 6,000 optimization iterations, namely that sometimes both the short- and long-term performance is improved, sometimes both are worsened, sometimes the short-term performance is improved but the long-term performance is worsened, etc.

For JDE the results are similarly mixed as shown in figure 4.10. On the Rastrigin problem JDE performs slightly worse at the 6,000 iterations mark when its behavioural parameters were specifically tuned to perform well on all 12 benchmark problems using 6,000 iterations instead of 60,000. JDE stagnates and is unable to optimize the Rastrigin problem, whereas using behavioural parameters tuned for 60,000 iterations it is able to find fitness values close to the optimal value of zero later in the optimization run. On the Schwefel1-2 problem there is a small improvement both before and after the 6,000 iterations mark, however, it would seem that there is just a lag between the two performance curves and from the quartiles the performance can be seen to be somewhat irregular and overlapping so it is hard to say that one choice of behavioural parameters is definitely better than the other on the Schwefel1-2 problem. On the Schwefel2-21 problem the parameters tuned for 6,000 iterations show a tiny improvement prior to that mark, but then seem to stagnate and the performance then becomes worse than for the parameters that were tuned for 60,000 iterations. On the Schwefel2-22 problem there is a slight performance improvement prior to the 6,000 iterations mark and that improvement becomes greater as optimization progresses. The results are similar on the other benchmark problems, namely that short- and long-term performance is sometimes improved and sometimes worsened, as was the case for DE/simple.

The cause of this might be that all 12 benchmark problems were used in the meta-optimization process and given the shorter optimization runs too much pressure was perhaps put on the DE variants to perform well on all 12 problems. To test this theory the behavioural parameters have instead been tuned for the four problems considered, namely the Rastrigin, Schwefel1-2, Schwefel2-21, and Schwefel2-22 problems. The parameters thus found are shown in table 4.8. Comparing these to the parameters tuned for all 12 benchmark problems as shown in table 4.7, the parameters for DE/simple can be seen to be rather similar while the JDE parameters are very different. The optimization performance for DE/simple is shown in figure 4.11 where it is compared to the performance of using the parameters that were tuned for all 12 benchmark problems. Since these parameters are rather similar it is no surprise that the optimization results are also similar, with the exception of the Schwefel1-2 problem where the mean progress has worsened. However, it can be seen from the quartiles that the reason for this worsening is due to only a few optimization runs performing more poorly and the performance is otherwise on par with before. For JDE the results are shown in figure 4.12 and the performance at the 6,000 iterations mark has worsened slightly for the Rastrigin problem, improved slightly for the Schwefel1-2 problem, and is about the same for the Schwefel2-21 and 2-22 problems. Interestingly, the mean performance on Rastrigin and Schwefel2-21 and 2-21 up until the 6,000 iterations mark is somewhat worse than when the behavioural parameters were tuned for all 12 benchmark problems. This must

be due to the inner workings of the JDE algorithm and is something that is not taken into account in the meta-optimization technique that was employed here, which merely tried to find behavioural parameters that improved performance at exactly the 6,000 iterations mark. The performance after this 6,000 iterations mark is improved towards the end on the Rastrigin problem and it is improved immediately after this mark on the Schwefel2-21 problem. On the Schwefel1-2 and Schwefel2-22 problems the performance is actually worsened significantly compared to before.

The conclusion is therefore that tuning the behavioural parameters for just these four problems instead of all twelve benchmark problems in an attempt to improve performance at the 6,000 iterations mark does not have any consistently improving effect. For the DE/simple variant the results were rather similar and for the JDE/rand/1/bin variant the results were sometimes better and sometimes worse. It therefore seems that 6,000 optimization iterations are perhaps too few iterations for the DE variants to work well without making performance concessions. It also seems that tuning the parameters for just 6,000 optimization iterations more often results in stagnation than using parameters that were tuned for 60,000 iterations. This implies that behavioural parameters cannot always be tuned for shorter optimization runs and still be expected to perform well for longer optimization runs, as was otherwise found to be the case for the meta-fitness landscapes plotted in chapter 3. For these reasons the remaining experiments will be conducted with 60,000 optimization iterations or more to be able to properly distinguish performance characteristics of optimizer variants.

4.4.5 Generalization Ability

It would be preferable to not have to tune the behavioural parameters of an optimization method for each new problem encountered. The ability of an optimization method to perform well on problems for which its behavioural parameters were not specifically tuned may be called its generalization ability. The purpose of this experiment is to establish the generalization ability of the DE/simple and JDE/rand/1/bin optimizer variants. Recall that JDE was devised specifically with the intent of alleviating parameter tuning by adapting its parameters during optimization and this experiment will seek to uncover whether JDE has any consistent advantage over DE/simple in this regard, as DE/simple keeps its behavioural parameters fixed during optimization. In order to test this, three different sets of benchmark problems will be used in meta-optimization and the performance on the remainder of the benchmark problems will then be studied. This is a simple form of cross-validation, see [122] for a survey of cross-validation techniques. The sets of benchmark problems are:

- Rosenbrock & Sphere, one hard and one easy problem.
- Rastrigin & Schwefel1-2, because DE/simple had difficulty optimizing Rastrigin and JDE had difficulty optimizing Schwefel1-2.

- QuarticNoise, Sphere & Step, because they are all unimodal problems with QuarticNoise having noise added and Step being a discontinuous version of Sphere. Parameters tuned for these simple problems may not generalize well to harder problems.

DE/simple

The behavioural parameters for DE/simple that were meta-optimized with regard to the three sets of benchmark problems are shown in table 4.9. The parameters that are most notably different from the parameters tuned for all 12 benchmark problems are the ones tuned for the QuarticNoise, Sphere & Step problems, which have significantly smaller population size NP and crossover probability CR. The parameters tuned for Rastrigin & Schwefel1-2 are very similar to those tuned for all 12 benchmark problems.

Using the parameters tuned for the Rosenbrock & Sphere problems gives the optimization results in figures 4.13 and 4.14 as well as the end results shown in table 4.11. The first thing to note concerns the mean fitness achieved after 60,000 iterations on the Rosenbrock and Sphere problems, which was the performance measure the behavioural parameters were being tuned for. The mean fitness achieved on Rosenbrock is similar to that achieved from using parameters that were tuned for all 12 benchmark problems, but the quartiles reveal that the new parameters yield better performance in many of the optimization runs. On the Sphere problem the mean fitness performance is improved significantly, but it was already so close to the optimal fitness of zero that it is only of interest in the sense that the optimum is now being approached quicker. Regarding the performance on the remaining problems for which the parameters were not specifically tuned, the performance is worsened somewhat on most of the other problems, perhaps with exception of the Schwefel1-2 problem where the quartiles reveal that better solutions are now found. Also, optimization now progresses a bit faster on the Schwefel2-22 problem.

Using the parameters tuned for the Rastrigin & Schwefel1-2 problems gives the optimization results in figures 4.15 and 4.16. The performance on Rastrigin seems to be unchanged and DE/simple still cannot optimize that problem even though its parameters were now tuned partially for that problem. On the Schwefel1-2 problem the mean performance is somewhat worse towards the end, but the quartiles reveal that better solutions are actually found. On the remaining benchmark problems the performance is comparable to when the behavioural parameters were tuned for all 12 benchmark problems, which is not a surprise since the behavioural parameters are so similar. The exception is the Schwefel2-22 problem on which the mean fitness is now worse but the quartiles reveal that it is due to occasional performance outliers.

Using the parameters tuned for the QuarticNoise, Sphere & Step problems gives the optimization results in figures 4.17 and 4.18. Performance is improved on the QuarticNoise and Step problems. The QuarticNoise problem is noisy and the fitness results are within the range of the noise and are therefore comparable. Performance on the Sphere problem is worsened slightly, but it is so

close to the optimal fitness value of zero that they too are deemed comparable. The overall sum of these is decreased, however, and as it is the measure that meta-optimization attempted to tune for, it explains why the Sphere result may have been allowed to get worse because the overall fitness sum has improved. On the remaining benchmark problems for which the behavioural parameters were not specifically tuned, performance has worsened on several problems including Rosenbrock, Schwefel1-2 and Schwefel2-22. Interestingly, performance has improved on the Rastrigin problem which DE/simple was previously having difficulty optimizing, although the optimum is still not quite located.

Overall it can be concluded that meta-optimizing the behavioural parameters of DE/simple with regard to these subsets of benchmark problems yields a comparable or slightly improved performance on the problems for which the parameters were tuned, while the performance was sometimes worse and sometimes better on the problems for which the parameters were not specifically tuned.

JDE/rand/1/bin

The behavioural parameters for JDE/rand/1/bin that were meta-optimized with regard to the three sets of benchmark problems are shown in table 4.10. Due to the opaque nature of the JDE algorithm and its parameters, only the population size NP will be commented here as it can be readily seen to be quite different, ranging from $NP = 14$ when tuned for the Rosenbrock & Sphere problems to $NP = 97$ when tuned for the QuarticNoise, Sphere & Step problems.

Using the parameters tuned for the Rosenbrock & Sphere problems gives the optimization results in figures 4.19 and 4.20. On the Rosenbrock and Sphere problems for which the parameters were tuned optimization progress is now faster, although the Rosenbrock end results are similar to before and the Sphere results were already so close to the optimum that the results should just be deemed comparable. On the remaining benchmark problems for which the JDE parameters were not specifically tuned there seems to be greater irregularity as shown by the quartiles. Take the Penalized1 and 2 problems, for example, where the mean performance is significantly worse but the quartiles show that significantly better solutions are indeed found. However, as these are all close to the optimal fitness value of zero the results on Penalized1 and 2 should also just be deemed comparable. Performance does appear to be considerably worsened on Rastrigin and Schwefel1-2, though.

Using the parameters tuned for the Rastrigin & Schwefel1-2 problems gives the optimization results in figures 4.21 and 4.22. On the Rastrigin problem the performance is similar up until the last 15000 optimization iterations where the mean performance is now worse and the quartiles reveal great irregularity with fewer good solutions being found. On the Schwefel1-2 problem there is a small improvement for the entire duration of optimization. Again, the explanation for why meta-optimization has found this choice of parameters that worsens the performance on one of the problems it was supposedly tuned for, while improving performance on the other problem, is due to the equal weighting of the

problems. From table 4.12 it can be seen that the sum of the fitness results when using the parameters that were tuned for all 12 benchmark problems is the mean fitness on Rastrigin plus the mean fitness on Schwefel1-2, that is, $0.02 + 10.38 = 10.40$, while it is $4.27 + 5.56 = 9.83$ when using the parameters tuned specifically for these two problems. Although these numbers are stochastic and may vary somewhat, it does explain why the meta-optimized parameters exhibit this tendency of sometimes worsening and sometimes improving individual benchmark results, as it is the overall performance sum that is sought improved. Concerning the performance on the benchmark problems for which these parameters were not specifically tuned there is clearly worse performance on Schwefel2-22, but otherwise the performance seems to be comparable taking the quartiles into account and the fact that fitness values close to optimal are often found.

Using the parameters tuned for the QuarticNoise, Sphere & Step problems gives the optimization results in figures 4.23 and 4.24. On the QuarticNoise problem there is a slight improvement in the entire optimization progress. On the Sphere problem there is improvement towards the end of the optimization run. On the Step problem there is a slight worsening due to the fact that one or a few of the optimization runs achieve sub-optimal results. Considering the proximity to optimal fitness values the performance is deemed comparable overall on the three problems for which the behavioural parameters were tuned. On the problems for which the parameters were not specifically tuned there is dramatically worse performance on the Schwefel1-2 problem and also somewhat worse on the Rastrigin and Schwefel2-21 problems. On the Rosenbrock and Schwefel2-22 problems the performance is comparable. Taking the irregularity of the results into account as well as their proximity to optimal fitness values, the performance is also somewhat comparable on the remaining problems, being a little better perhaps on the Ackley problem whilst a little worse on the Penalized1 and 2 problems.

The overall conclusion for the generalization ability of JDE/rand/1/bin is similar to that of DE/simple above, namely that by tuning the behavioural parameters for only a subset of the 12 benchmark problems, performance is sometimes improved on the remaining problems for which the parameters were not specifically tuned and sometimes the performance is worsened. So there does not appear to be any consistent advantage to using JDE over DE/simple in terms of the generalization ability. This is important because an assumed generalization ability was part of the justification for introducing the more complicated JDE variant in the first place.

4.4.6 Specialization Ability

The purpose of these experiments is to establish how well the DE/simple and JDE/rand/1/bin optimizers are able to specialize to certain problems. This is useful when one needs to optimize a number of problems that are closely related but where the standard choice of behavioural parameters do not perform well.

It has been observed in the previous experiments that DE/simple is unable

to optimize Ackley and Rastrigin with the behavioural parameters tested. Table 4.13 shows the behavioural parameters for DE/simple tuned to perform well on the Ackley and Rastrigin problems individually. These are almost the inverse of the parameters found to work well on all 12 benchmark problems in an overall manner, which explains why poor performance was observed on those two problems. The population size NP has decreased significantly, the crossover probability CR has decreased to be almost zero, its lower limit, whereas before it was closer to its upper limit of one. The differential weight F has increased to twice its previous value, or more. The optimization progress of using these specialized behavioural parameters is shown in figure 4.25 and can be seen to be a great improvement as solutions close to the optimal fitness values are now being found.

For the JDE/rand/1/bin variant the previous experiments showed performance problems on the Rosenbrock and Schwefel1-2 problems. The behavioural parameters tuned for these two problems individually are shown in table 4.14 but are again difficult to interpret due to their opaque nature, although it can be noted that the population size NP appears to be somewhat similar to that of the parameters tuned for all 12 benchmark problems. The optimization progress of using these specialized behavioural parameters is shown in figure 4.26 and is not as impressive an improvement as for DE/simple. Significant performance improvement on the Rosenbrock problem occurs only at the end where near-optimal fitness values are achieved on occasion. On the Schwefel1-2 problem better results are obtained but there is also a much increased irregularity.

From these experiments it would therefore seem that JDE does not specialize quite as well as DE/simple does, although one should be wary of concluding too much from just a few experiments as the reverse may be true on other optimization problems.

4.4.7 Long Optimization Runs

This experiment will study the performance of DE/simple and JDE/rand/1/bin using longer optimization runs. This is only useful in practise when the fitness function is fast to evaluate. Many studies in the literature report results for such long optimization runs, see e.g. [36] [25], and it is apparently believed that adaptive parameter schemes need additional iterations for the adaptation to become effective.

It has been observed in the previous experiments that DE/simple had difficulty optimizing the Ackley and Rastrigin problems when its behavioural parameters were not tuned specifically for those problems. Whether this was caused by too short optimization runs will now be tested. There are no hand-tuned behavioural parameters available for DE/simple so the parameters meta-optimized for all 12 benchmark problems when allowed 60,000 optimization iterations will be used, see table 4.9. Figure 4.27 shows that the optimization progress stagnates early during optimization and it therefore seems reasonable to conclude that DE/simple is unable to optimize the Ackley and Rastrigin problems using

this choice of behavioural parameters, seemingly no matter how many optimization iterations are allowed.

Concerning JDE/rand/1/bin it has been observed that especially the Rosenbrock and Schwefel1-2 problems were difficult for JDE to optimize. Using the standard behavioural parameters for JDE the Rastrigin and Schwefel2-21 problems were also not quite optimized within 60,000 iterations, see table 4.4. Figure 4.28 shows the optimization progress when allowing 10 times as many iterations. First consider the use of standard parameters where it can be seen that near-optimal fitness values are indeed found for the Rastrigin, Rosenbrock and Schwefel1-2 problems when allowing more iterations. Performance on the Schwefel2-21 problem is irregular and apparently stagnates before the optimal fitness value of zero is quite reached. Now compare this to the use of behavioural parameters that were tuned for all 12 benchmark problems using 60,000 optimization iterations, also depicted in figure 4.28, from which it can be seen that optimization generally progresses faster. On the Rastrigin problem both choices of parameters find near-optimal fitness values and are therefore deemed comparable. For the Rosenbrock problem the performance is comparable until around 250,000 iterations where the standard parameters start to perform better. On the Schwefel1-2 problem optimization is improved overall. On the Schwefel2-21 problem optimization initially progresses faster and the mean fitness is improved throughout, but the quartiles reveal that slightly better fitness values are found by use of the standard parameters. It can therefore be concluded that JDE is capable of optimizing most of the 12 benchmark problems with the standard choice of behavioural parameters, provided enough iterations are being allowed. Tuning the parameters for use with 60,000 optimization iterations sometimes leads to better and sometimes worse results when allowing 10 times as many iterations.

The behavioural parameters can be tuned specifically for these longer optimization runs but due to the computational time involved only four benchmark problems are being used: Ackley, Rastrigin, Rosenbrock and Schwefel1-2, that is, some of the benchmark problems these DE variants had most difficulties optimizing. The behavioural parameters thus found are shown in table 4.15 and the optimization progress from their usage are shown in figures 4.29 and 4.30. The first thing to note is that DE/simple is now able to optimize Ackley on occasion but still cannot optimize Rastrigin. The performance of DE/simple has worsened on the Rosenbrock and especially the Schwefel1-2 problems on which it previously performed well, see e.g. table 4.6, and for which the behavioural parameters were now tuned. This seems peculiar and the reason is perhaps that the inability to perform well on all four problems using one choice of behavioural parameters has led to these performance concessions so as to improve the fitness overall. Regarding JDE, it is now able to optimize and find near-optimal fitness values for all 12 benchmark problems, even for the Schwefel2-21 problem which was not quite optimized using either standard parameters or parameters that were tuned for shorter optimization runs, see figure 4.28. Comparing DE/simple to JDE it can be seen from figures 4.29 and 4.30 that in spite of the deficiencies of DE/simple it does hold an advantage over JDE on some of the problems in

that optimization progresses faster using DE/simple, see the Penalized1 and 2, Schwefel2-21, Schwefel2-22, Sphere and Step problems.

The conclusion is that JDE seems to hold an advantage over DE/simple when longer optimization runs are allowed, in that JDE is able to optimize all 12 benchmark problems and find near-optimal fitness values, while DE/simple apparently cannot have its behavioural parameters tuned so as to work well for all problems simultaneously. The cause of this is unknown although one could speculate as to whether the so-called adaptive schemes of JDE did indeed become effective when allowed more optimization iterations. It should be noted, however, that JDE had its performance further improved by meta-optimizing its behavioural parameters for these longer optimization runs and the technique of meta-optimization is therefore still useful.

4.4.8 Weights in Meta-Optimization

It has been observed in the experiments above that meta-optimization may improve the overall fitness while worsening performance on individual problems, the reason being that equal weights were being used on all the problems during tuning of behavioural parameters. The weights are denoted w_i in the meta-optimization algorithm depicted in figure 3.3 and have been set to $w_i = 1$ in the previous experiments. It was noted in chapter 3 that choosing the meta-fitness weights is not a trivial task and this experiment will test various combinations of weights in meta-optimization.

Recall that DE/simple had performance difficulties on the Ackley and Rastrigin problems except when the behavioural parameters were specifically tuned for those problems individually, see section 4.4.6. Table 4.16 shows the behavioural parameters of DE/simple when tuned for the Ackley, Rastrigin, Rosenbrock and Schwefel1-2 problems using various weights in the meta-fitness measure. Setting the meta-fitness weight to 10000 for one problem, e.g. Rastrigin, and 1 for the remaining problems, is done in an attempt to make the performance on Rastrigin dominate the meta-fitness measure so that the behavioural parameters are tuned primarily for that problem. Since the benchmark problems have vastly different fitness ranges the weights have been chosen rather large. The DE/simple parameters that were tuned for these four problems with equal weights seem to be dominated by the Schwefel1-2 problem, because the parameters are rather similar to the ones tuned for the Schwefel1-2 being weighted heavily. The parameters tuned for the Ackley and Rastrigin problems have a much smaller population size NP and crossover probability CR, and a higher differential weight F. Table 4.17 shows the optimization end results of using these behavioural parameters. Compared to the results of using parameters tuned with equal weights for these four problems, it is evident that a heavy weight on one of the problems causes the tuned parameters to perform better on that one problem at the cost of degraded performance on one or more of the other problems. It seems that performance on Ackley and Rastrigin are somewhat correlated, in the sense that parameters tuned for Ackley being heavily weighted also yield fair results on Rastrigin and vice versa. On the other

hand, parameters tuned for either or both of Ackley and Rastrigin being heavily weighted yield poor performance on the Rosenbrock and Schwefel1-2 problems, and vice versa. So it would seem that good performance cannot be achieved on all four problems using one common choice of parameters for DE/simple.

The conclusion is that meta-fitness weights may be useful when an optimization method cannot have its behavioural parameters tuned to perform well on all the problems considered, because there is some inability of the optimizer to perform well on all the problems with only one choice of behavioural parameters. The practitioner can then set weights to be used in meta-optimization so as to determine the mutual importance of the problems, thus ensuring good performance on certain problems while perhaps worse performance on the other problems. The choice of weights will depend on the practitioner's needs, the problems considered, as well as the capabilities of the optimization method whose parameters are to be tuned. The practitioner may therefore have to do some experimentation with weights in meta-optimization.

4.4.9 Parameter Consistency

From all the meta-optimization experiments above only the best found behavioural parameters were being used. Because the meta-fitness measure used in guiding the tuning process is stochastic by nature it raises the question whether the best found parameters were indeed accurate and representative of good choices of parameters, or if the parameters were merely deemed best due to stochastic irregularity and were in fact inferior. To gain additional confidence in the parameters found through meta-optimization a list of the best found parameters can be studied for consistency. Although this has not been described previously it was actually done in all of the experiments above. Table 4.18 shows the ten best sets of parameters found for the meta-optimization experiments using all 12 benchmark problems and 60,000 optimization iterations, that is, the experiments from sections 4.4.2 and 4.4.3. These ten sets of parameters are taken from all five meta-optimization runs conducted on each optimizer variant and are therefore not necessarily found in just one meta-optimization run. For most of these parameters there are clear and consistent tendencies, for example, there is a tendency for a population size for DE/simple of $NP \simeq 180$ and for JDE of $NP \simeq 40$. The basic DE/rand/1/bin seems to perform worse with increasing NP and the Dither variant does not seem to be that affected by the choice of NP. There are other interesting parameter tendencies, for instance, DE/rand/1/bin also seems to perform worse with increasing crossover probability CR (and decreasing differential weight F), while the DE/simple, Dither, Jitter and JDE variants seem to work best with high CR. It should be noted that these observations are made for behavioural parameters that were tuned for particular optimization scenarios and may not translate to other optimization problems and run-lengths.

Overall the meta-optimized behavioural parameters of an optimizer seem to be consistent, especially taking into account the small number of iterations performed by the meta-optimizer to traverse the search-space of behavioural

parameters. The behavioural parameters are also consistent in terms of the meta-fitness measure which does not fluctuate wildly due to stochastic noise. This consistency and stability adds to the credibility of the meta-optimization technique for finding good choices behavioural parameters.

4.4.10 Time Usage

The time usage for some of the previous meta-optimization experiments have been listed in table 4.19. All experiments were conducted on an Intel Pentium M 1.5 GHz laptop computer, which is already several years old, using an implementation in the C# programming language. The experimental settings were the LUS method as meta-optimizer, five meta-optimization runs per experiment, each consisting of 20 times the number of behavioural parameters to be tuned, that is, 60 iterations of the meta-optimizer when tuning the parameters of DE/simple which has 3 behavioural parameters, and the optimizer was run 50 times on each benchmark problem using the designated number of optimization iterations.

It should be noted that different combinations of benchmark problems may require different amounts of time in meta-optimization, for example, tuning DE/simple for the Rosenbrock and Sphere problems when allowing 6,000 optimization iterations takes roughly 41 minutes, while it takes roughly 79 minutes using the Rastrigin and Schwefel1-2 problems. So in this comparison the same problem configurations have been used for both DE/simple and JDE. It can be seen from table 4.19 that DE/simple is generally faster to tune than JDE, often by a factor three but sometimes less. Although JDE does have three times as many behavioural parameters as DE/simple it does not always take three times as much time to tune the parameters. The reason is the use of Preemptive Fitness Evaluation as described in section 3.4 which may alleviate time usage somewhat differently for optimizers.

4.5 Summary

The first overall conclusion from this chapter regards the best performing DE variant, where various experiments were conducted to establish the ability of the DE variants to perform well in terms of the generalization and specialization ability, for short and long optimization runs, etc. The only scenario in which the complicated JDE/rand/1/bin variant appeared to have a consistent advantage over the much simpler DE/simple variant was for long optimization runs. The reason seems to be that DE/simple could not have its behavioural parameters tuned so as to work well on all benchmark problems simultaneously. In this regard it would seem that the so-called adaptive behavioural parameters of JDE were indeed successful, although it is perhaps ironic that the performance of JDE was improved by tuning its behavioural parameters as well. There were also a number of scenarios in which DE/simple held an advantage over JDE, in particular the optimization progress was faster for DE/simple on some

problems and its behavioural parameters seemed to specialize better when tuned for benchmark problems individually. Due to fewer behavioural parameters of DE/simple it was also significantly faster to tune than JDE. Whether JDE or DE/simple is best will therefore depend on the practitioner's needs.

The second conclusion regards the usefulness of meta-optimization, where it could be argued that the performance improvement resulting from tuning the behavioural parameters in an offline manner, is not justified by the large number of additional optimization iterations needed to perform such tuning. However, it should be kept in mind that the behavioural parameters of an optimization method must be tuned at least once before the method is published, whether this is done by a human researcher or by a computer. Freeing up human resources by having a computer perform automated parameter tuning seems desirable and it may even lead to greater insight as it makes it easier for the human researcher to study an optimizer's performance in a variety of scenarios, such as generalization and specialization ability, short and long optimization runs, weighting of the performance on different problems, and so on.

Basic	$NP \in \{4, \cdots, 200\}$	$CR \in [0, 1]$	$F \in [0, 2]$
Dither	$NP \in \{4, \cdots, 200\}$	$CR \in [0, 1]$	$F_{mid} \in [0, 2]$ $F_{range} \in [0, 3]$
Jitter	$NP \in \{4, \cdots, 200\}$	$CR \in [0, 1]$	$F_{mid} \in [0, 2]$ $F_{range} \in [0, 3]$
JDE	$NP \in \{4, \cdots, 200\}$	$CR_{init} \in [0, 1]$ $CR_l \in [0, 1]$ $CR_u \in [0, 1]$ $\tau_{CR} \in [0, 1]$	$F_{init} \in [0, 2]$ $F_l \in [0, 2]$ $F_u \in [0, 2]$ $\tau_F \in [0, 1]$

Table 4.1: Boundaries for the parameter search-spaces of DE/rand/1/bin variants as used in all the meta-optimization experiments.

Basic	$NP = 50$	$CR = 0.9$	$F = 0.6$
Dither	$NP = 50$	$CR = 0.9$	$F_{mid} = 0.75$ $F_{range} = 0.25$
Jitter	$NP = 50$	$CR = 0.9$	$F_{mid} = 0.5$ $F_{range} = 0.0005$
JDE	$NP = 100$	$CR_{init} = 0.9$ $CR_l = 0$ $CR_u = 1$ $\tau_{CR} = 0.1$	$F_{init} = 0.5$ $F_l = 0.1$ $F_u = 0.9$ $\tau_F = 0.1$

Table 4.2: Standard behavioural parameters for DE/rand/1/bin variants. Optimization results are found in table 4.4 and figures 4.1 and 4.2.

Basic	$NP = 19$	$CR = 0.1220$	$F = 0.4983$
Dither	$NP = 102$	$CR = 0.9637$	$F_{mid} = 0.7876$ $F_{range} = 0.7292$
Jitter	$NP = 58$	$CR = 0.9048$	$F_{mid} = 0.3989$ $F_{range} = 0.3426$
JDE	$NP = 32$	$CR_{init} = 0.0947$ $CR_l = 0.9166$ $CR_u = 0.0834$ $\tau_{CR} = 0.0180$	$F_{init} = 0.6068$ $F_l = 0.3462$ $F_u = 1.1388$ $\tau_F = 0.0561$

Table 4.3: Behavioural parameters for DE/rand/1/bin variants meta-optimized for all **12 benchmark problems** in 30 dimensions each and optimization runlengths of **60,000** iterations. Optimization results are found in table 4.5 and figures 4.3 and 4.4.

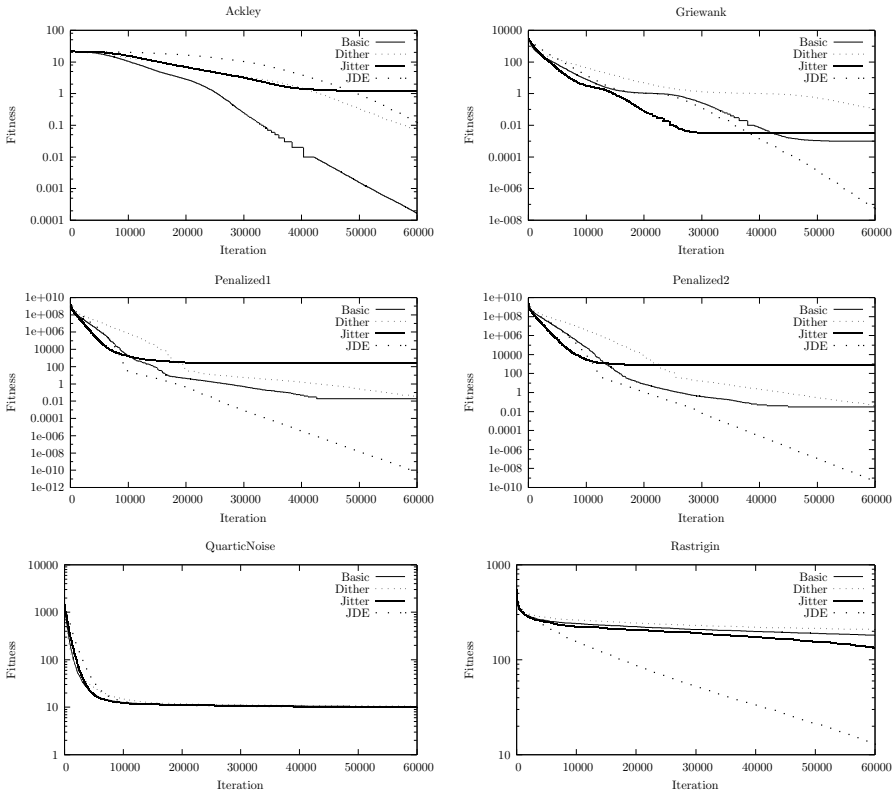

Figure 4.1: Comparison of optimization progress for DE/rand/1/bin variants using the behavioural parameters from table 4.2 which are **standard** in the literature. Plots show the mean fitness achieved over 50 optimization runs.

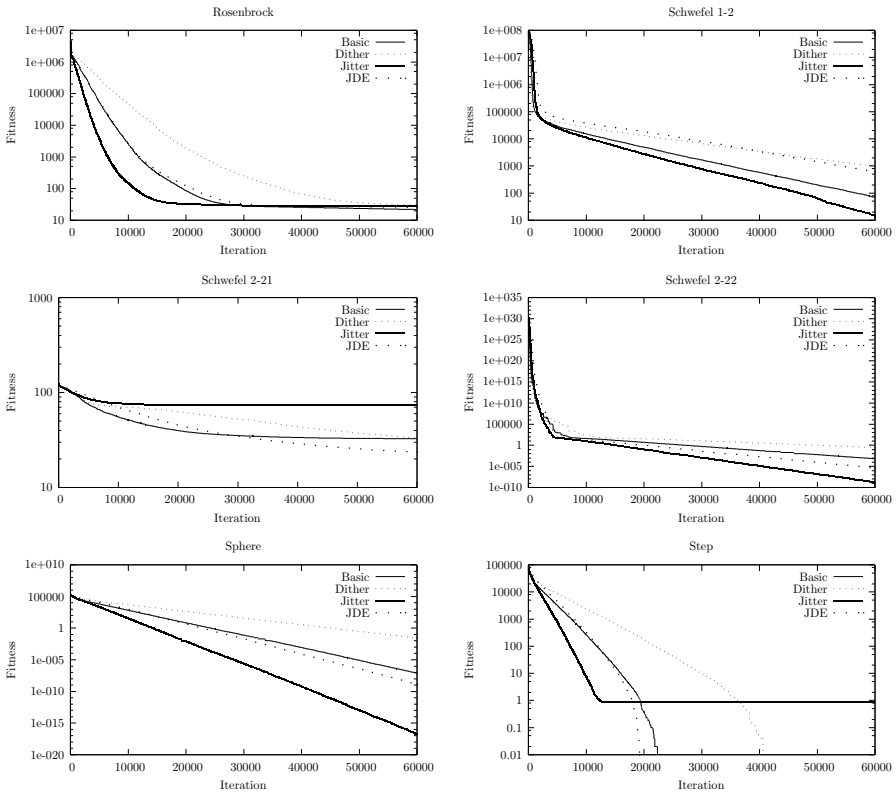

Figure 4.2: Comparison of optimization progress for DE/rand/1/bin variants using the behavioural parameters from table 4.2 which are **standard** in the literature. Plots show the mean fitness achieved over 50 optimization runs.

	Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
Basic	Ackley	1.67e-4	8.16e-5	4.13e-5	1.03e-4	1.39e-4	2.06e-4	3.67e-4
	Griewank	9.86e-4	3.19e-3	3.4e-8	1.58e-7	3e-7	7.52e-7	0.02
	Penalized1	0.02	0.12	8.05e-10	2e-8	9.14e-8	3.98e-7	0.83
	Penalized2	0.03	0.22	3.54e-9	5.4e-8	1.35e-7	4.35e-7	1.6
	QuarticNoise	10.34	0.94	9.06	9.97	10.29	10.51	15.99
	Rastrigin	181.1	21.91	120	161.32	187.01	197.45	210.62
	Rosenbrock	21.72	1.4	17.46	21.17	21.79	22.44	26.52
	Schwefel1-2	70.79	38.83	24.45	40.16	61.42	87.64	216.44
	Schwefel2-21	32.68	14.08	4.24	22.41	31.67	42.76	61.85
	Schwefel2-22	6.59e-4	3.69e-4	1.57e-4	3.84e-4	5.47e-4	9.25e-4	1.79e-3
	Sphere	7.97e-8	6.52e-8	1.35e-8	3.29e-8	6.54e-8	9.95e-8	3.5e-7
	Step	0	0	0	0	0	0	0
Dither	Ackley	0.07	0.03	0.03	0.05	0.06	0.08	0.21
	Griewank	0.1	0.11	0.02	0.04	0.06	0.12	0.52
	Penalized1	0.03	0.09	2.67e-4	2.21e-3	4.72e-3	0.01	0.48
	Penalized2	0.04	0.06	3.32e-3	0.01	0.02	0.05	0.32
	QuarticNoise	10.57	0.47	9.23	10.21	10.59	10.84	11.41
	Rastrigin	208.84	16.31	155.77	200.21	209.47	222.22	239.76
	Rosenbrock	30.28	13.84	25.16	26.89	27.57	28.14	98.95
	Schwefel1-2	978.9	442.25	453.43	678.32	918.49	1109	3308
	Schwefel2-21	33.06	22.26	4.71	13.86	25.42	54.32	75
	Schwefel2-22	0.3	0.16	0.07	0.19	0.26	0.35	0.93
	Sphere	0.03	0.01	7.09e-3	0.02	0.02	0.03	0.07
	Step	0	0	0	0	0	0	0
Jitter	Ackley	1.25	4.94	1.15e-9	5.56e-9	2.54e-8	1.93e-6	20.95
	Griewank	3.35e-3	6.68e-3	0	0	0	7.4e-3	0.03
	Penalized1	228.36	1346	4.15e-19	5.17e-14	2.36e-9	0.04	9452
	Penalized2	753.61	4955	1.75e-19	1.9e-16	1.26e-12	2.51e-7	35364
	QuarticNoise	10.11	0.43	9.02	9.86	10.09	10.43	11.07
	Rastrigin	133.36	39.52	17.7	106.62	138.74	167.61	200.44
	Rosenbrock	28.04	13.86	21.01	24.59	25.3	25.87	110.12
	Schwefel1-2	14.9	41.78	0.56	2.66	5.4	8.55	292.78
	Schwefel2-21	73.73	8.34	52.78	70.9	73.67	78.23	88.78
	Schwefel2-22	1.71e-9	2.18e-9	9.36e-11	6.07e-10	9.6e-10	1.99e-9	1.1e-8
	Sphere	1.93e-17	4.71e-17	4.41e-19	3.15e-18	5.94e-18	1.21e-17	3.21e-16
	Step	0.9	4.11	0	0	0	0	25
JDE	Ackley	0.12	0.65	3.41e-5	3.52e-4	4.69e-3	0.02	4.59
	Griewank	5.22e-8	1.83e-7	6.2e-10	4.4e-9	8.16e-9	1.69e-8	1.07e-6
	Penalized1	5.89e-11	4.74e-11	7.94e-12	2.66e-11	4.33e-11	8.19e-11	2.6e-10
	Penalized2	4.36e-10	3.5e-10	5.43e-11	2.03e-10	3.61e-10	5.73e-10	2.1e-9
	QuarticNoise	9.84	0.42	8.59	9.52	9.87	10.18	10.67
	Rastrigin	12.91	3.46	4.55	10.92	12.63	15.36	20.62
	Rosenbrock	25.7	7.71	23.43	24.39	24.65	24.97	79.58
	Schwefel1-2	594.1	431.73	121.53	264.18	485.15	709.89	2009
	Schwefel2-21	23.51	11.83	4.23	14.1	21.73	34.77	50.11
	Schwefel2-22	3.99e-6	1.77e-6	1.08e-6	2.74e-6	3.54e-6	5.29e-6	9.2e-6
	Sphere	1.33e-9	1.23e-9	1.3e-10	6.5e-10	9.05e-10	1.51e-9	6.84e-9
	Step	0	0	0	0	0	0	0

Table 4.4: Optimization end results for DE/rand/1/bin variants using the behavioural parameters from table 4.2 which are **standard** in the literature. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 60,000.

	Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
Basic	Ackley	1.21	3.82	3.15e-14	9.16e-10	5.8e-8	5.3e-6	17.06
	Griewank	0.38	2.44	0	0	0	0	17.39
	Penalized1	0.02	0.1	1.57e-32	1.57e-32	1.57e-32	1.57e-32	0.62
	Penalized2	1.88e-32	2.85e-32	1.35e-32	1.35e-32	1.35e-32	1.35e-32	2.13e-31
	QuarticNoise	12.66	1.43	10.1	11.77	12.65	13.32	18.05
	Rastrigin	17.81	7	4.97	11.94	16.91	21.89	33.83
	Rosenbrock	33.33	20.43	9.44	23.35	25.59	27.03	83.81
	Schwefel1-2	2262	1313	678.23	1487	1833	2823	7374
	Schwefel2-21	75.11	4.5	67.92	72.15	74.73	77.62	93.11
	Schwefel2-22	1.78e-17	8.7e-17	0	0	0	0	4.44e-16
	Sphere	1.14e-44	6.48e-44	7.71e-48	4.92e-47	1.07e-46	3.12e-46	4.59e-43
	Step	0.08	0.56	0	0	0	0	4
Dither	Ackley	0.36	0.61	7.11e-3	0.01	0.03	0.11	2.22
	Griewank	0.02	0.01	6.26e-4	6.91e-3	0.01	0.02	0.07
	Penalized1	5.76	14.19	0.03	1.46	3.02	5.2	102.67
	Penalized2	5.2	15.56	6.86e-4	1.24	2.72	4.44	112.6
	QuarticNoise	10.06	0.58	8.97	9.67	10.07	10.36	11.95
	Rastrigin	52.88	22.65	21.6	35.42	45.04	67.9	107.96
	Rosenbrock	28.08	7.81	23.83	25.93	27.08	27.87	81.87
	Schwefel1-2	99.25	61.83	19.62	48.46	89.34	120.41	264.79
	Schwefel2-21	35.87	12.51	10.51	26.25	33.25	44.15	69.7
	Schwefel2-22	0.01	6.54e-3	4.24e-3	6.61e-3	9.17e-3	0.01	0.04
	Sphere	3.41e-3	3.32e-3	3.67e-4	1.67e-3	2.52e-3	3.44e-3	0.02
	Step	0.74	1.78	0	0	0	1	9
Jitter	Ackley	2.61	6.78	4.74e-10	5.83e-8	1.57e-6	4.22e-4	20.99
	Griewank	3.44e-3	8.61e-3	0	0	0	1.11e-16	0.05
	Penalized1	9012	58619	7.63e-18	3.69e-14	1.26e-10	7.09e-4	418108
	Penalized2	72.65	507.99	1.31e-19	2.54e-18	1.16e-16	9.7e-11	3629
	QuarticNoise	10.18	1.14	8.77	9.76	10	10.23	15.89
	Rastrigin	84.81	23.52	30.92	68.1	88.19	102.63	126.46
	Rosenbrock	28.34	10.19	22.65	25.94	26.22	26.6	78.83
	Schwefel1-2	80.89	93.68	9.22	30.51	52.9	86.35	609.98
	Schwefel2-21	80.65	6.66	65.28	75.41	80.49	87.12	91.25
	Schwefel2-22	4.17e-10	9e-10	4.17e-11	1.12e-10	1.84e-10	3.82e-10	6.32e-9
	Sphere	2.82e-18	4.08e-18	9.84e-20	4.6e-19	8.12e-19	2.93e-18	1.82e-17
	Step	0.08	0.27	0	0	0	0	1
JDE	Ackley	1.1e-5	2.74e-5	7.86e-9	6.34e-7	1.92e-6	6.15e-6	1.77e-4
	Griewank	1.04e-3	2.88e-3	1.11e-16	1.29e-14	9.4e-14	6.3e-13	0.01
	Penalized1	5e-17	2.17e-16	5.89e-21	5.55e-19	3.01e-18	1.23e-17	1.45e-15
	Penalized2	1.67e-13	1.15e-12	3.97e-20	7.78e-18	3.01e-17	2.08e-15	8.24e-12
	QuarticNoise	10.2	0.56	8.65	9.86	10.28	10.54	11.5
	Rastrigin	0.02	0.12	1.15e-8	1.73e-5	2.47e-4	5.41e-3	0.88
	Rosenbrock	24.26	1.07	21.89	23.78	24.28	24.83	27.13
	Schwefel1-2	10.38	13.26	0.63	3.2	5.96	12.45	77.18
	Schwefel2-21	14.66	10.56	1.2	6.37	13.5	18.71	49.51
	Schwefel2-22	3.2e-9	1.16e-8	2.05e-12	2.59e-11	1.34e-10	8.26e-10	7.98e-8
	Sphere	3.13e-11	2.19e-10	4.08e-18	4.61e-16	2.47e-15	2.43e-14	1.56e-9
	Step	0	0	0	0	0	0	0

Table 4.5: Optimization end results for DE/rand/1/bin variants using the behavioural parameters from table 4.3 which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 60,000.

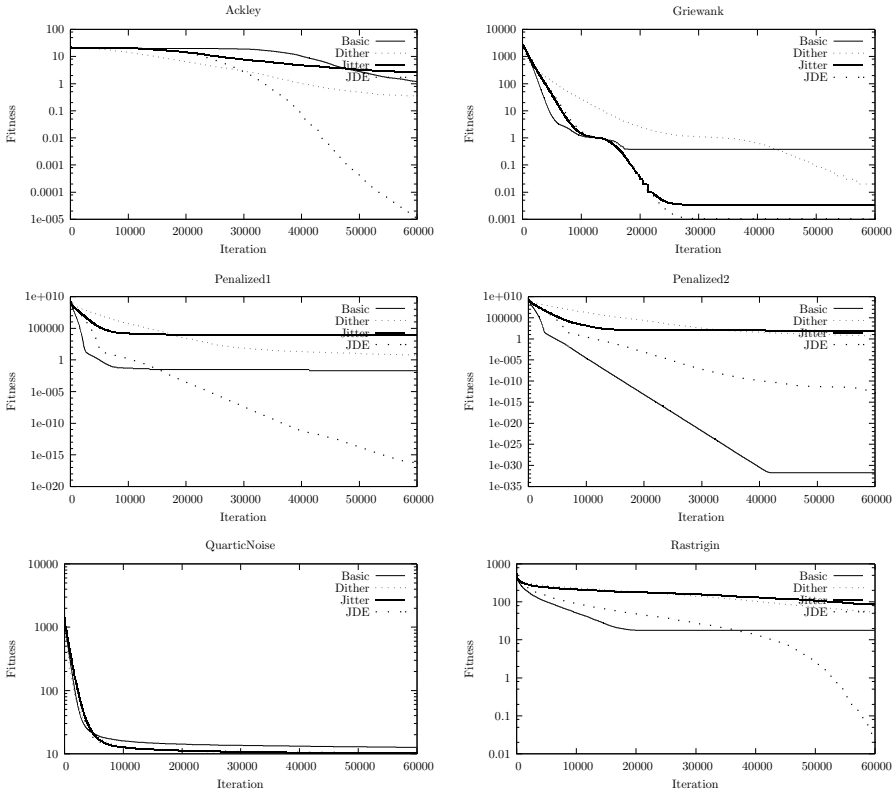

Figure 4.3: Comparison of optimization progress for DE/rand/1/bin variants using the behavioural parameters from table 4.3 which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs.

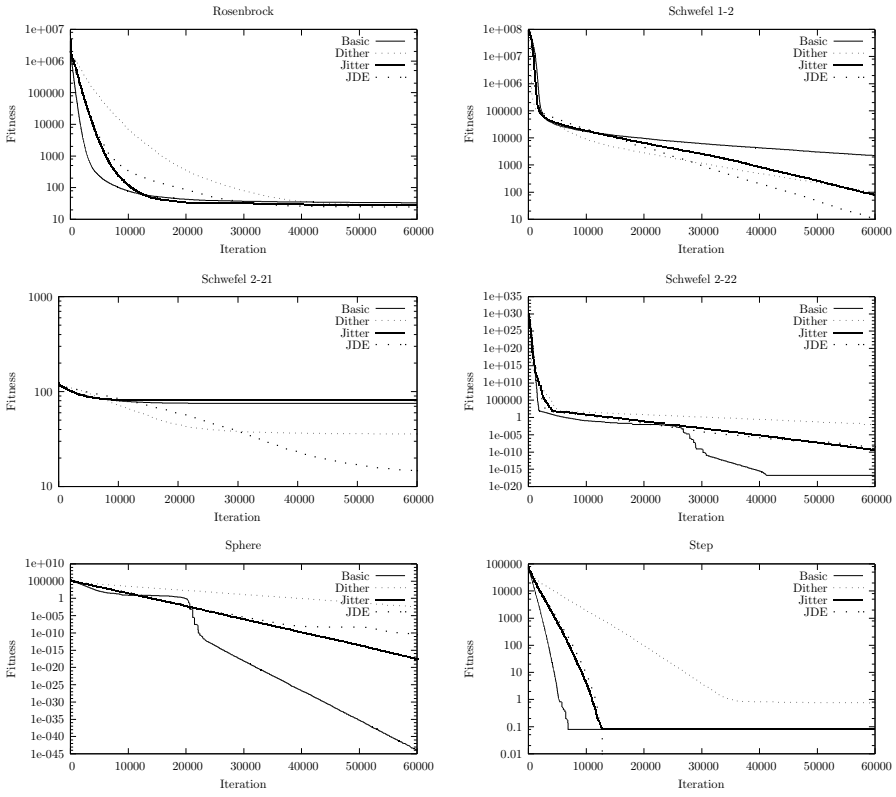

Figure 4.4: Comparison of optimization progress for DE/rand/1/bin variants using the behavioural parameters from table 4.3 which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs.

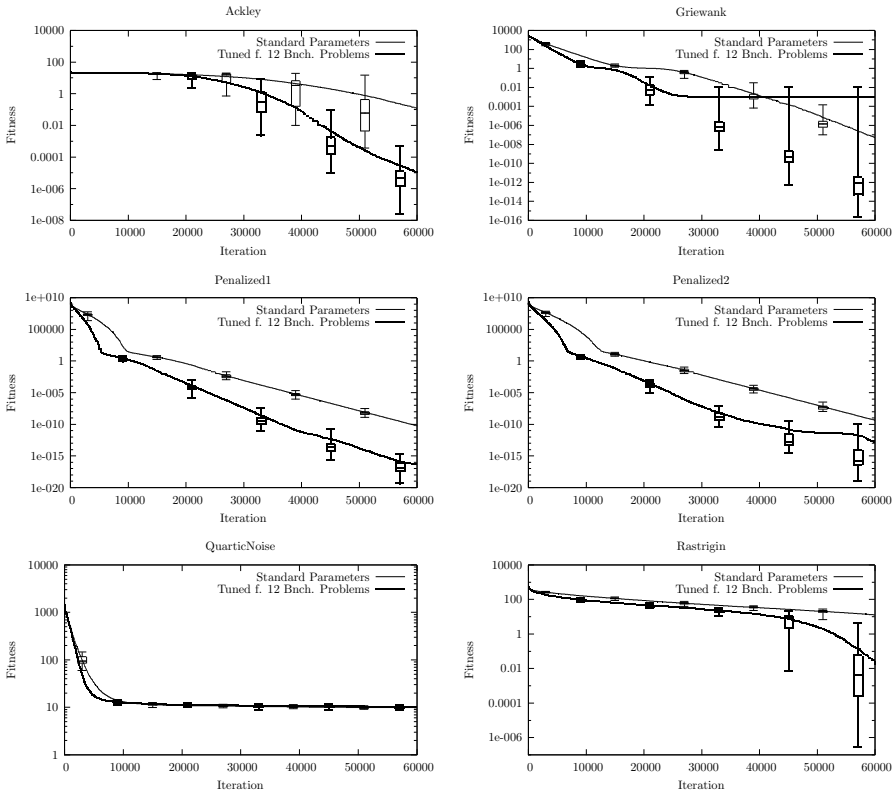

Figure 4.5: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from tables 4.2 and 4.3, which were respectively standard in the literature and meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

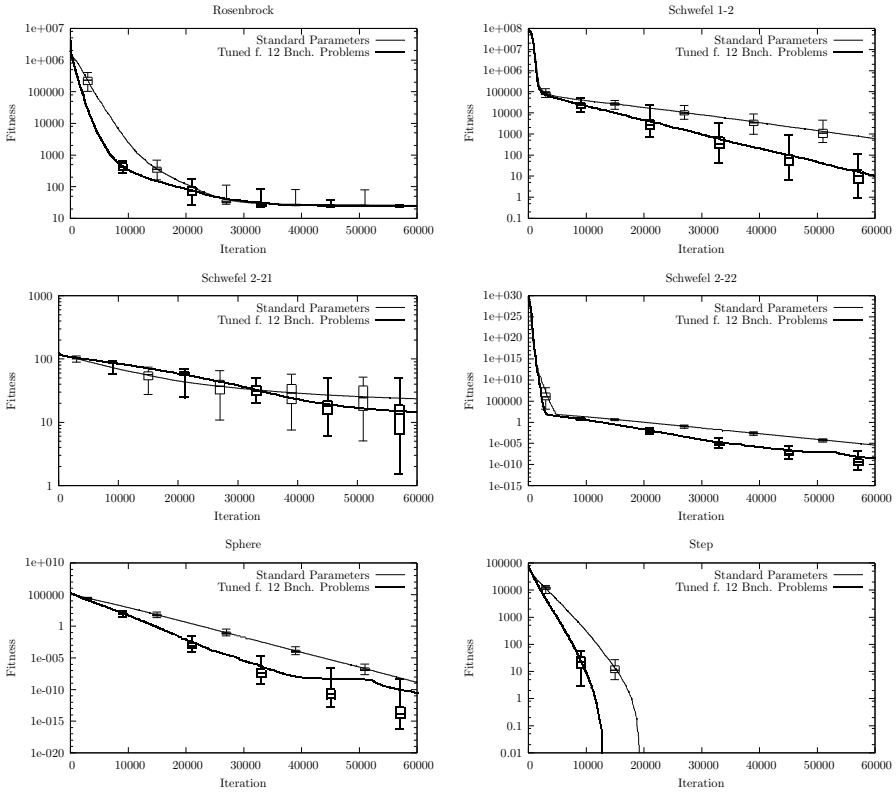

Figure 4.6: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from tables 4.2 and 4.3, which were respectively standard in the literature and meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

	Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
DE/simple	Ackley	20.57	0.39	19.93	19.98	20.79	20.87	20.97
	Griewank	8.22e-3	0.01	0	0	7.4e-3	0.01	0.04
	Penalized1	0.89	1.64	1.57e-32	2.25e-31	0.21	0.94	9.19
	Penalized2	0.08	0.25	1.6e-32	7.27e-32	0.01	0.02	1.59
	QuarticNoise	13.06	2.08	9.57	11.61	12.83	14.54	18.49
	Rastrigin	159.04	27.93	102.48	137.28	155.71	184.01	204.9
	Rosenbrock	1.12	1.79	7.96e-9	2.63e-4	8.1e-4	3.99	4.01
	Schwefel1-2	0.6	3.92	2.33e-5	2.55e-4	2.77e-3	7.87e-3	28.01
	Schwefel2-21	2.36e-3	3.04e-3	1.63e-4	7.48e-4	1.39e-3	2.86e-3	0.02
	Schwefel2-22	0.5	2.45	1.78e-15	8.44e-15	1.54e-12	6.3e-10	12.5
	Sphere	3.07e-39	4.45e-39	1.09e-40	4.24e-40	9.9e-40	3.21e-39	1.72e-38
	Step	2.18	2.46	0	1	1	2	10
JDE	Ackley	1.1e-5	2.74e-5	7.86e-9	6.34e-7	1.92e-6	6.15e-6	1.77e-4
	Griewank	1.04e-3	2.88e-3	1.11e-16	1.29e-14	9.4e-14	6.3e-13	0.01
	Penalized1	5e-17	2.17e-16	5.89e-21	5.55e-19	3.01e-18	1.23e-17	1.45e-15
	Penalized2	1.67e-13	1.15e-12	3.97e-20	7.78e-18	3.01e-17	2.08e-15	8.24e-12
	QuarticNoise	10.2	0.56	8.65	9.86	10.28	10.54	11.5
	Rastrigin	0.02	0.12	1.15e-8	1.73e-5	2.47e-4	5.41e-3	0.88
	Rosenbrock	24.26	1.07	21.89	23.78	24.28	24.83	27.13
	Schwefel1-2	10.38	13.26	0.63	3.2	5.96	12.45	77.18
	Schwefel2-21	14.66	10.56	1.2	6.37	13.5	18.71	49.51
	Schwefel2-22	3.2e-9	1.16e-8	2.05e-12	2.59e-11	1.34e-10	8.26e-10	7.98e-8
	Sphere	3.13e-11	2.19e-10	4.08e-18	4.61e-16	2.47e-15	2.43e-14	1.56e-9
	Step	0	0	0	0	0	0	0

Table 4.6: Optimization end results for DE/simple and JDE/rand/1/bin using the behavioural parameters for DE/simple from Eq.(4.3) and for JDE from table 4.3, which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 60,000. Results for JDE are reprinted from table 4.5.

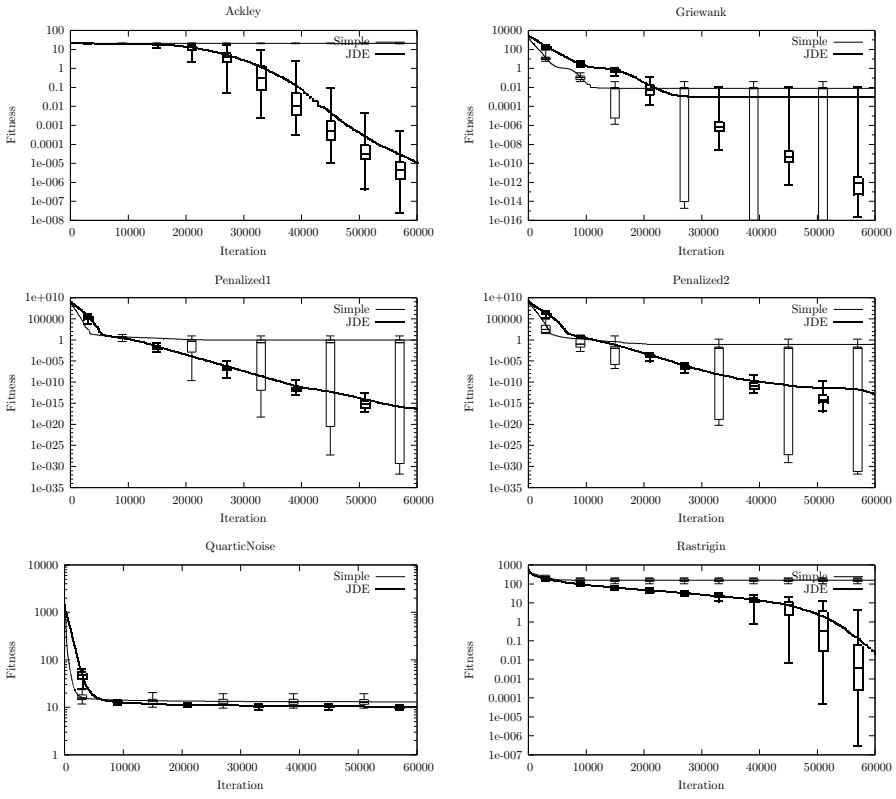

Figure 4.7: Comparison of optimization progress for DE/simple and JDE using the behavioural parameters for DE/simple from Eq.(4.3) and for JDE from table 4.7, which were meta-optimized for all **12 benchmark problems** using **6,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

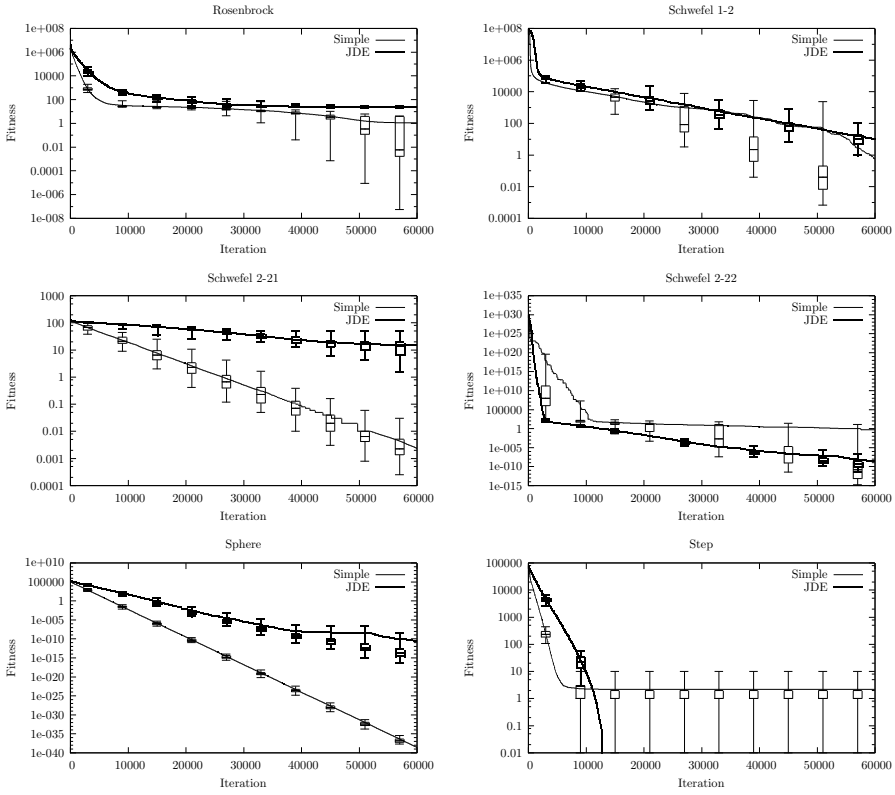

Figure 4.8: Comparison of optimization progress for DE/simple and JDE using the behavioural parameters for DE/simple from Eq.(4.3) and for JDE from table 4.7, which were meta-optimized for all **12 benchmark problems** using **6,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

DE/simple	$NP = 136$	$CR = 0.9813$			$F = 0.2790$		
JDE	$NP = 28$	CR_{init}	=	0.5	F_{init}	=	0.4304
		CR_l	=	0.4826	F_l	=	0.3009
		CR_u	=	0.5174	F_u	=	0.6392
		τ_{CR}	=	0.1805	τ_F	=	0.6449

Table 4.7: Behavioural parameters for DE/simple and JDE/rand/1/bin that are meta-optimized for all **12 benchmark problems** in 30 dimensions each and optimization run-lengths of **6,000** iterations. Optimization results are found in figures 4.9 and 4.10.

DE/simple	$NP = 103$	$CR = 0.9794$			$F = 0.3976$		
JDE	$NP = 77$	CR_{init}	=	0.3911	F_{init}	=	1.8074
		CR_l	=	0.9893	F_l	=	0.2417
		CR_u	=	0.0107	F_u	=	1.0375
		τ_{CR}	=	0.8041	τ_F	=	0.5336

Table 4.8: Behavioural parameters for DE/simple and JDE/rand/1/bin that are meta-optimized for the **Rastrigin, Schwefel1-2, Schwefel2-21, Schwefel2-22** problems in 30 dimensions each and optimization run-lengths of **6,000** iterations. Optimization results are found in figures 4.11 and 4.12.

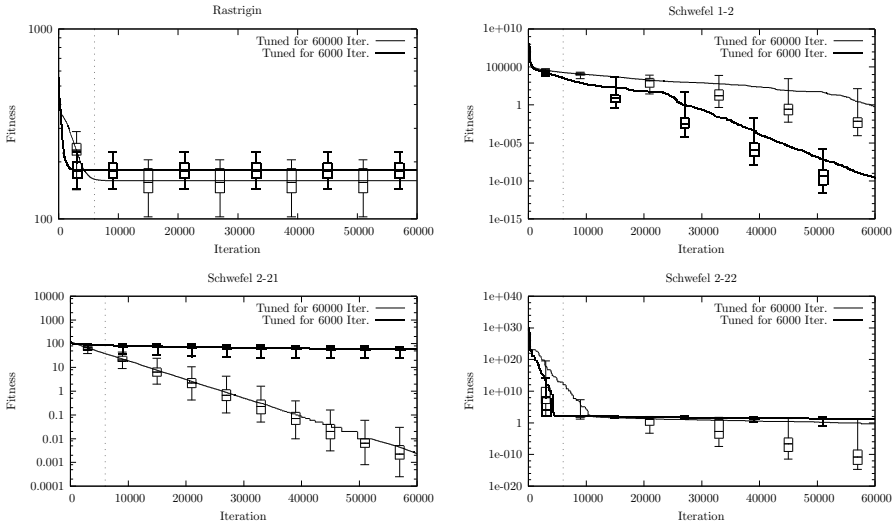

Figure 4.9: Comparison of optimization progress for **DE/simple** using the behavioural parameters from Eq.(4.3) and table 4.7, which were meta-optimized for all **12 benchmark problems** using **60,000** and **6,000** fitness evaluations respectively. Plots show the mean fitness achieved over 50 optimization runs with quartiles at intervals. Dotted line shows 6,000 fitness evaluations.

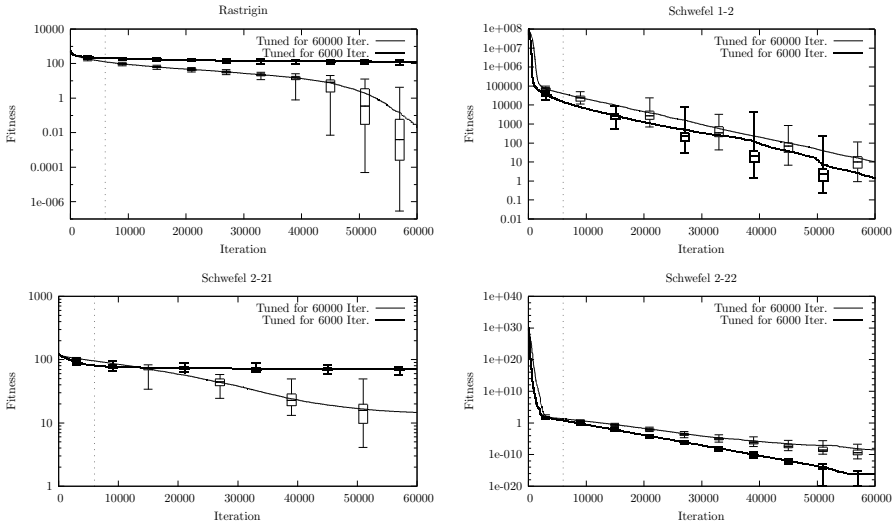

Figure 4.10: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from tables 4.3 and 4.7, which were meta-optimized for all **12 benchmark problems** using **60,000** and **6,000** fitness evaluations respectively. Plots show the mean fitness achieved over 50 optimization runs with quartiles at intervals. Dotted line shows 6,000 fitness evaluations.

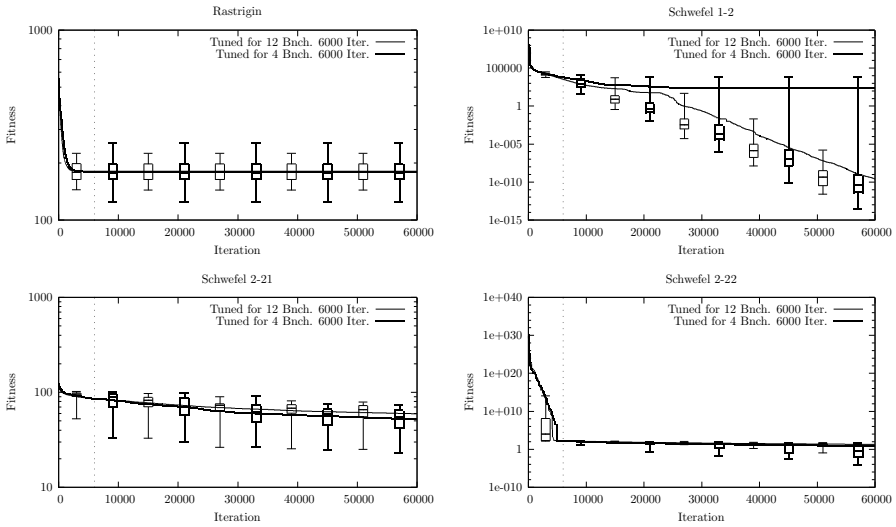

Figure 4.11: Comparison of optimization progress for **DE/simple** using the behavioural parameters from tables 4.7 and 4.8, which were meta-optimized for respectively all **12 benchmark problems** and the **Rastrigin, Schwefel1-2, Schwefel2-21, Schwefel2-22** problems using **6,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs with quartiles at intervals. Dotted line shows 6,000 fitness evaluations.

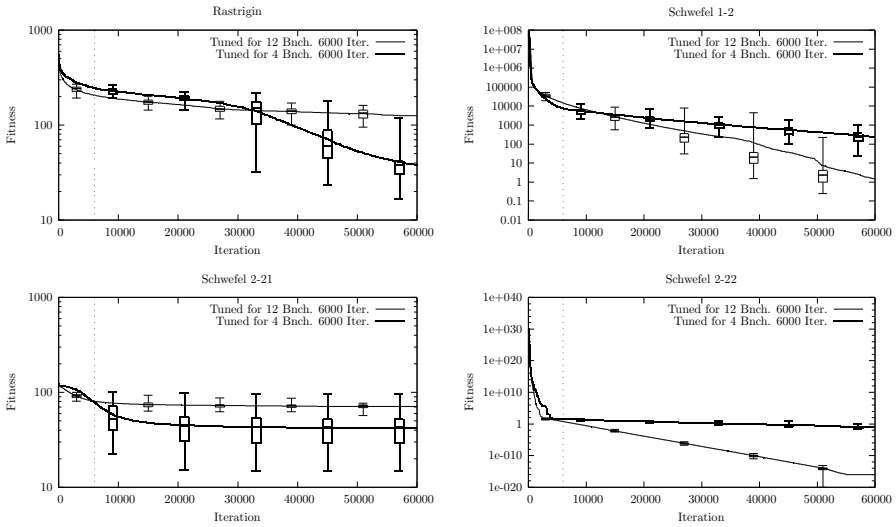

Figure 4.12: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from tables 4.7 and 4.8, which were meta-optimized for respectively all **12 benchmark problems** and the **Rastrigin, Schwefel1-2, Schwefel2-21, Schwefel2-22** problems using **6,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs with quartiles at intervals. Dotted line shows 6,000 fitness evaluations.

Problems	NP	CR	F
All Benchmark Problems	186	0.8493	0.4818
Rosenbrock & Sphere	126	0.9211	0.4027
Rastrigin & Schwefel1-2	185	0.8932	0.5125
QuarticNoise, Sphere & Step	106	0.3345	0.5860

Table 4.9: Behavioural parameters for **DE/simple** that are meta-optimized for various combinations of benchmark problems in 30 dimensions each and optimization run-lengths of **60,000** iterations. Optimization results are found in figures 4.13-4.18 and table 4.11.

Problems	NP	CR			F		
All Benchmark Problems	32	CR_{init}	=	0.0947	F_{init}	=	0.6068
		CR_l	=	0.9166	F_l	=	0.3462
		CR_u	=	0.0834	F_u	=	1.1388
		τ_{CR}	=	0.0180	τ_F	=	0.0561
Rosenbrock & Sphere	14	CR_{init}	=	0.1943	F_{init}	=	0.2091
		CR_l	=	0.0676	F_l	=	0.6697
		CR_u	=	0.6439	F_u	=	0.0962
		τ_{CR}	=	0.6275	τ_F	=	0.2369
Rastrigin & Schwefel1-2	40	CR_{init}	=	0.7029	F_{init}	=	1.8855
		CR_l	=	0.3101	F_l	=	0.1030
		CR_u	=	0.6899	F_u	=	0.9366
		τ_{CR}	=	0.0909	τ_F	=	0.2843
QuarticNoise, Sphere & Step	97	CR_{init}	=	0.8331	F_{init}	=	1.8255
		CR_l	=	0.7161	F_l	=	0.0437
		CR_u	=	0.1705	F_u	=	1.1422
		τ_{CR}	=	0.9808	τ_F	=	0.2514

Table 4.10: Behavioural parameters for **JDE/rand/1/bin** that are meta-optimized for various combinations of benchmark problems in 30 dimensions each and optimization run-lengths of **60,000** iterations. Optimization results are found in figures 4.19-4.24 and table 4.12.

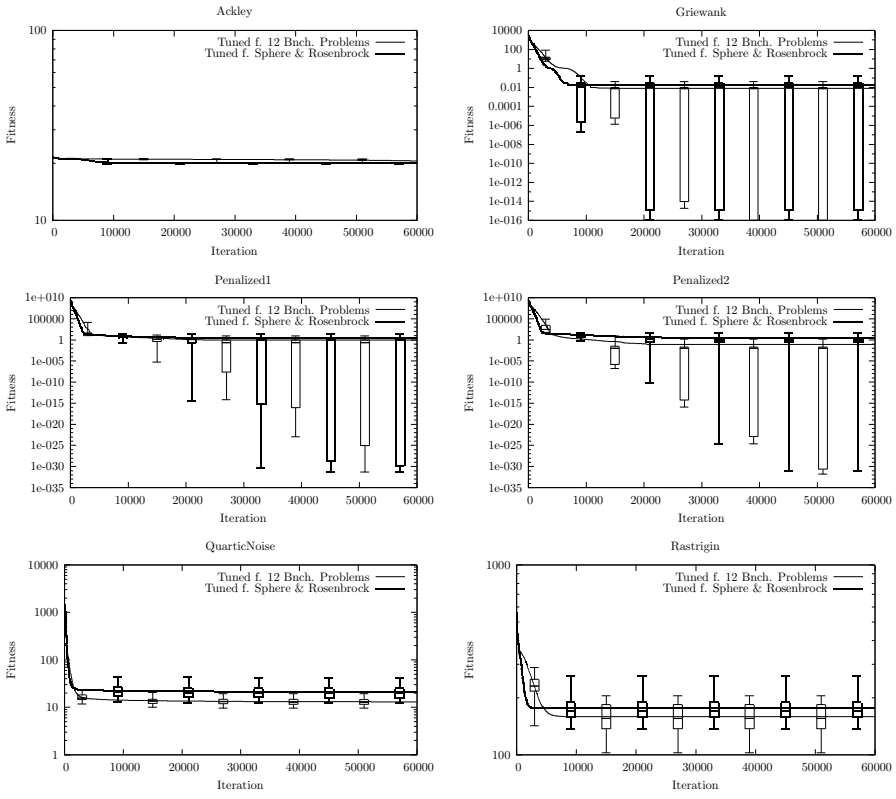

Figure 4.13: Comparison of optimization progress for **DE/simple** using the behavioural parameters from table 4.9, which were meta-optimized for respectively all **12 benchmark problems** and **Rosenbrock & Sphere** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

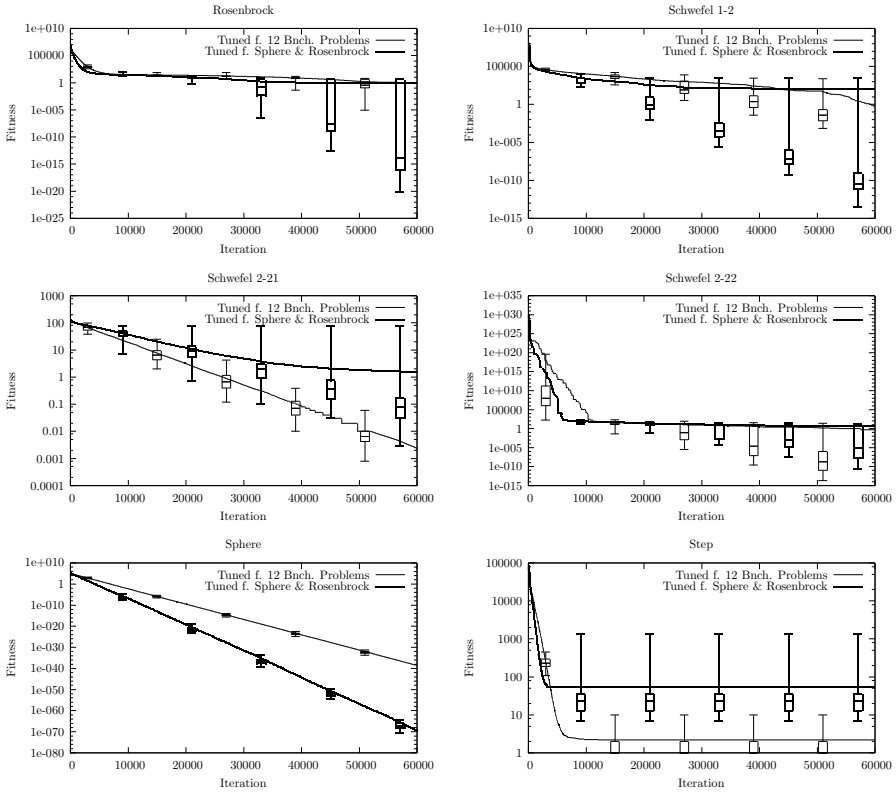

Figure 4.14: Comparison of optimization progress for **DE/simple** using the behavioural parameters from table 4.9, which were meta-optimized for respectively all **12 benchmark problems** and **Rosenbrock & Sphere** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

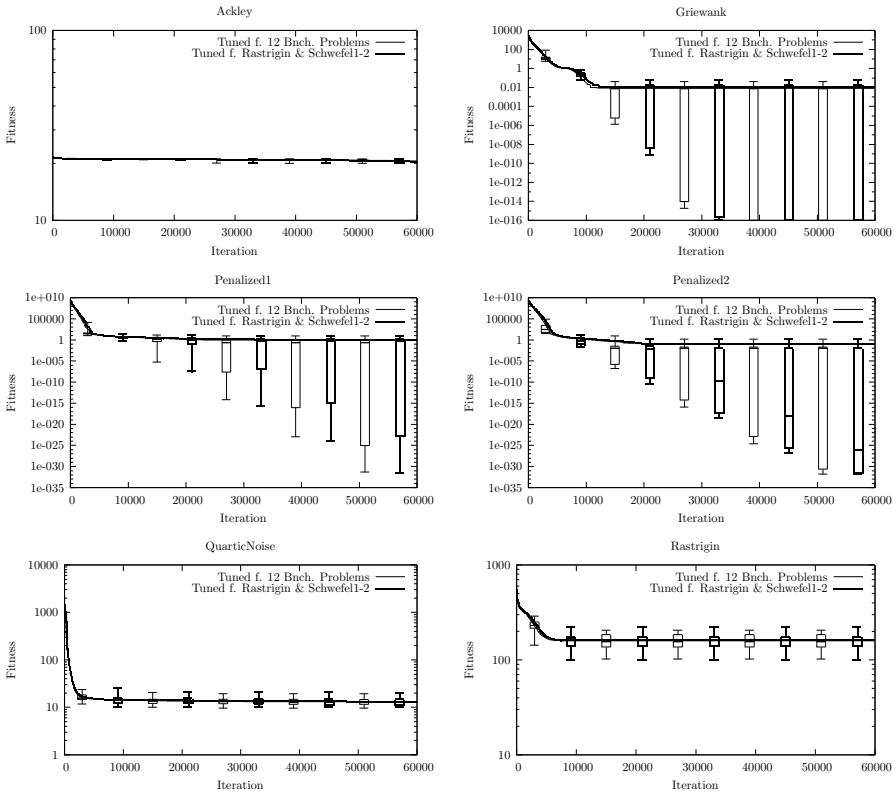

Figure 4.15: Comparison of optimization progress for **DE/simple** using the behavioural parameters from table 4.9, which were meta-optimized for respectively all **12 benchmark problems** and **Rastrigin & Schwefel1-2** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

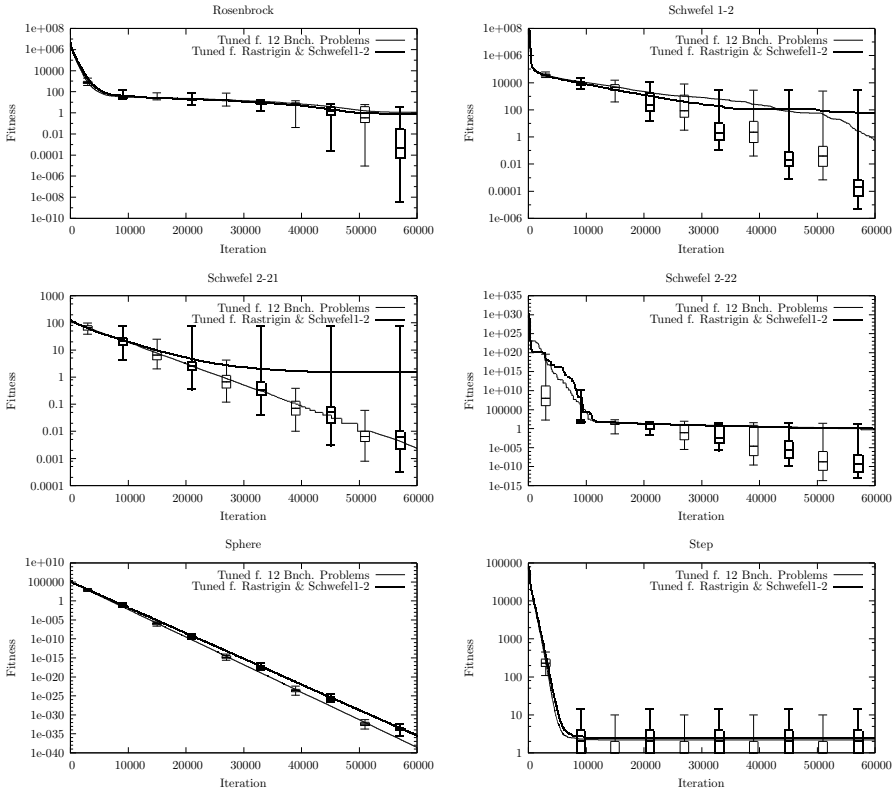

Figure 4.16: Comparison of optimization progress for **DE/simple** using the behavioural parameters from table 4.9, which were meta-optimized for respectively all **12 benchmark problems** and **Rastrigin & Schwefel1-2** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

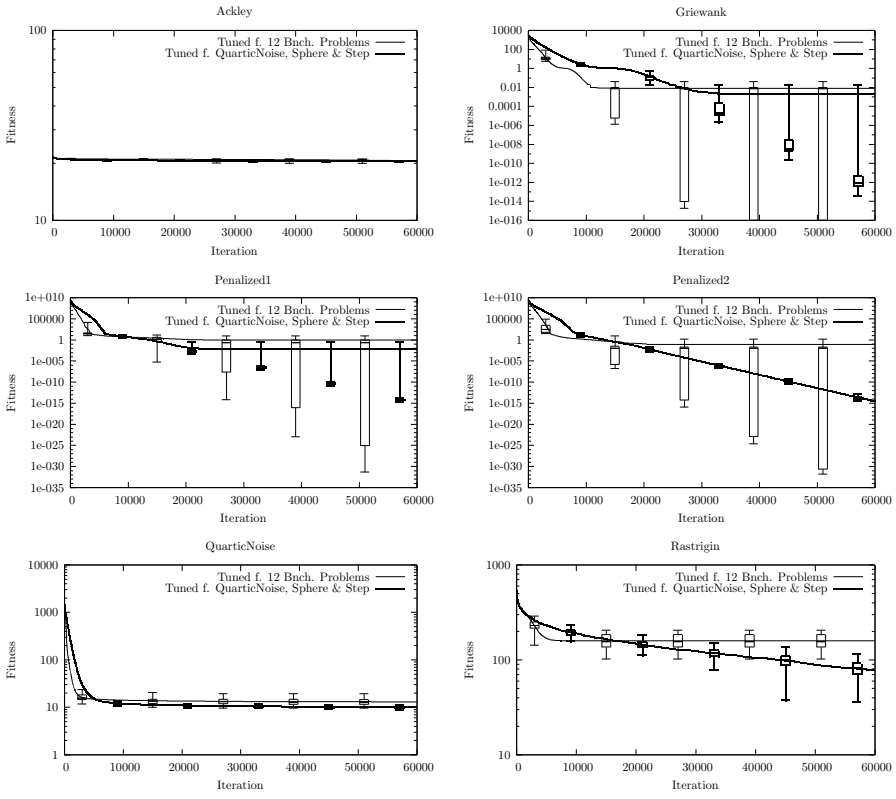

Figure 4.17: Comparison of optimization progress for **DE/simple** using the behavioural parameters from table 4.9, which were meta-optimized for respectively all **12 benchmark problems** and **QuarticNoise, Sphere & Step** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

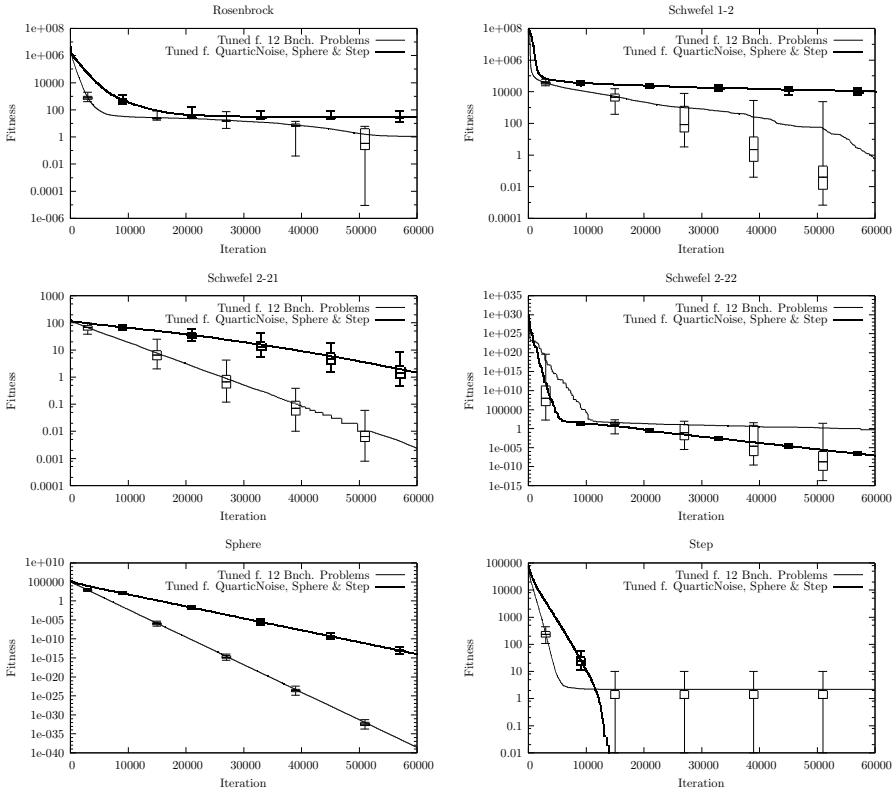

Figure 4.18: Comparison of optimization progress for **DE/simple** using the behavioural parameters from table 4.9, which were meta-optimized for respectively all **12 benchmark problems** and **QuarticNoise, Sphere & Step** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

	Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
All Benchmark Problems	Ackley	20.57	0.39	19.93	19.98	20.79	20.87	20.97
	Griewank	8.22e-3	0.01	0	0	7.4e-3	0.01	0.04
	Penalized1	0.89	1.64	1.57e-32	2.25e-31	0.21	0.94	9.19
	Penalized2	0.08	0.25	1.6e-32	7.27e-32	0.01	0.02	1.59
	QuarticNoise	13.06	2.08	9.57	11.61	12.83	14.54	18.49
	Rastrigin	159.04	27.93	102.48	137.28	155.71	184.01	204.9
	Rosenbrock	1.12	1.79	7.96e-9	2.63e-4	8.1e-4	3.99	4.01
	Schwefel1-2	0.6	3.92	2.33e-5	2.55e-4	2.77e-3	7.87e-3	28.01
	Schwefel2-21	2.36e-3	3.04e-3	1.63e-4	7.48e-4	1.39e-3	2.86e-3	0.02
	Schwefel2-22	0.5	2.45	1.78e-15	8.44e-15	1.54e-12	6.3e-10	12.5
	Sphere	3.07e-39	4.45e-39	1.09e-40	4.24e-40	9.9e-40	3.21e-39	1.72e-38
	Step	2.18	2.46	0	1	1	2	10
Rosenbrock & Sphere	Ackley	19.95	7.86e-3	19.93	19.94	19.95	19.95	19.96
	Griewank	0.02	0.03	1.11e-16	1.33e-15	0.01	0.03	0.15
	Penalized1	3.09	5.75	4.15e-32	1.77e-30	1.09	3.24	30.95
	Penalized2	3.04	7.41	9.24e-32	0.31	0.91	3.02	48.24
	QuarticNoise	21.25	6.86	12.07	16.11	20.19	25.42	41.53
	Rastrigin	175.55	25.59	137.3	159.16	170.62	189.52	260.08
	Rosenbrock	1.12	1.79	1.45e-22	3.43e-18	4.19e-16	3.99	3.99
	Schwefel1-2	112.5	551.14	5.45e-15	7.42e-13	4.07e-12	1.51e-10	2813
	Schwefel2-21	1.59	10.49	1.12e-3	0.02	0.06	0.11	75
	Schwefel2-22	4.15	6.07	2.52e-11	3.07e-9	3.47e-6	12.5	20
	Sphere	4.19e-70	1.18e-69	1.17e-75	1.02e-72	6.79e-72	1.14e-70	6.58e-69
	Step	53.66	179.92	7	13	23	36	1306
Rastrigin & Schwefel1-2	Ackley	20.4	0.4	19.94	19.96	20.28	20.81	20.98
	Griewank	0.01	0.01	0	1.11e-16	9.86e-3	0.02	0.06
	Penalized1	1.12	1.8	2.09e-32	1.29e-25	0.52	1.44	9.99
	Penalized2	0.08	0.26	1.35e-32	1.35e-32	9.2e-29	0.01	1.58
	QuarticNoise	13.14	2.36	10.22	11.35	12.7	14.76	20.59
	Rastrigin	160.86	30.91	99.47	139.27	164.12	175.08	222.3
	Rosenbrock	0.8	1.59	5.7e-10	5.81e-6	4.09e-5	1.27e-3	3.99
	Schwefel1-2	57.03	393.68	1.78e-6	1.38e-5	5.67e-5	2.6e-4	2813
	Schwefel2-21	1.51	10.5	2.42e-4	1.39e-3	3.87e-3	7.3e-3	75
	Schwefel2-22	1	3.39	2e-14	3.72e-13	7.16e-11	2.85e-8	12.5
	Sphere	3.58e-36	5.12e-36	3.55e-38	6.17e-37	2.11e-36	4.2e-36	2.7e-35
	Step	2.44	2.78	0	1	2	4	14
QuarticNoise, Sphere & Step	Ackley	20.47	0.05	20.35	20.43	20.47	20.5	20.6
	Griewank	2.12e-3	4.67e-3	5.44e-15	3.85e-14	8.68e-14	3.61e-13	0.02
	Penalized1	8.29e-3	0.06	1.68e-16	6.71e-16	9.38e-16	1.99e-15	0.41
	Penalized2	2.66e-15	2.37e-15	3.68e-16	1.1e-15	1.77e-15	3.21e-15	1.16e-14
	QuarticNoise	10.04	0.47	8.76	9.8	10.01	10.3	11.21
	Rastrigin	77.17	16.91	35.82	67.28	78.14	86.45	113.01
	Rosenbrock	28.58	15.04	13.28	23.95	24.09	24.31	81.64
	Schwefel1-2	10351	2546	5689	8636	10023	11780	17073
	Schwefel2-21	1.48	1.23	0.34	0.74	1.06	1.99	7.13
	Schwefel2-22	8e-8	5.06e-8	2.75e-8	4.95e-8	6.57e-8	9.8e-8	3.26e-7
	Sphere	9.71e-15	9.27e-15	1.33e-15	5.22e-15	7.56e-15	1.17e-14	6.33e-14
	Step	0	0	0	0	0	0	0

Table 4.11: Optimization end results for **DE/simple** using the behavioural parameters from table 4.9 which were meta-optimized for various combinations of benchmark problems using **60,000** fitness evaluations. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 60,000.

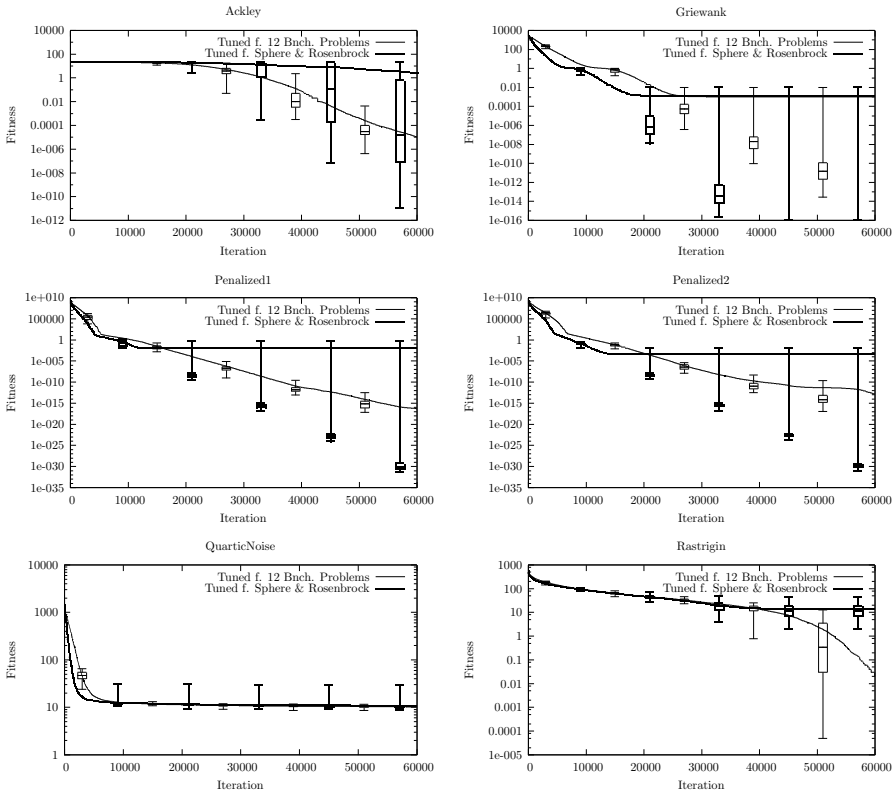

Figure 4.19: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from table 4.10, which were meta-optimized for respectively all **12 benchmark problems** and **Rosenbrock & Sphere** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

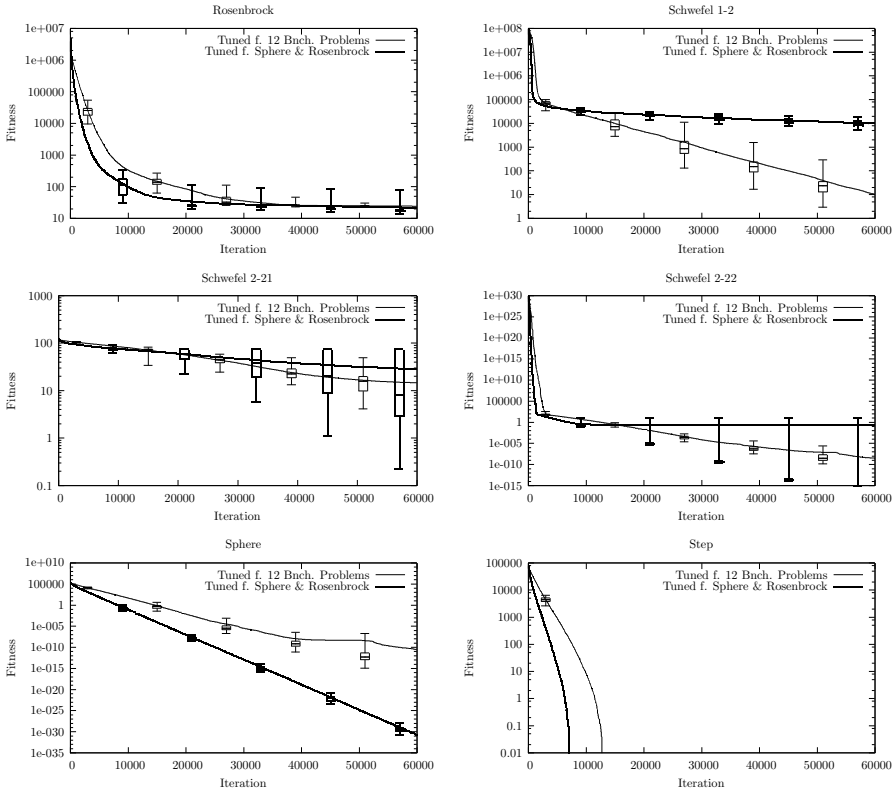

Figure 4.20: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from table 4.10, which were meta-optimized for respectively all **12 benchmark problems** and **Rosenbrock & Sphere** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

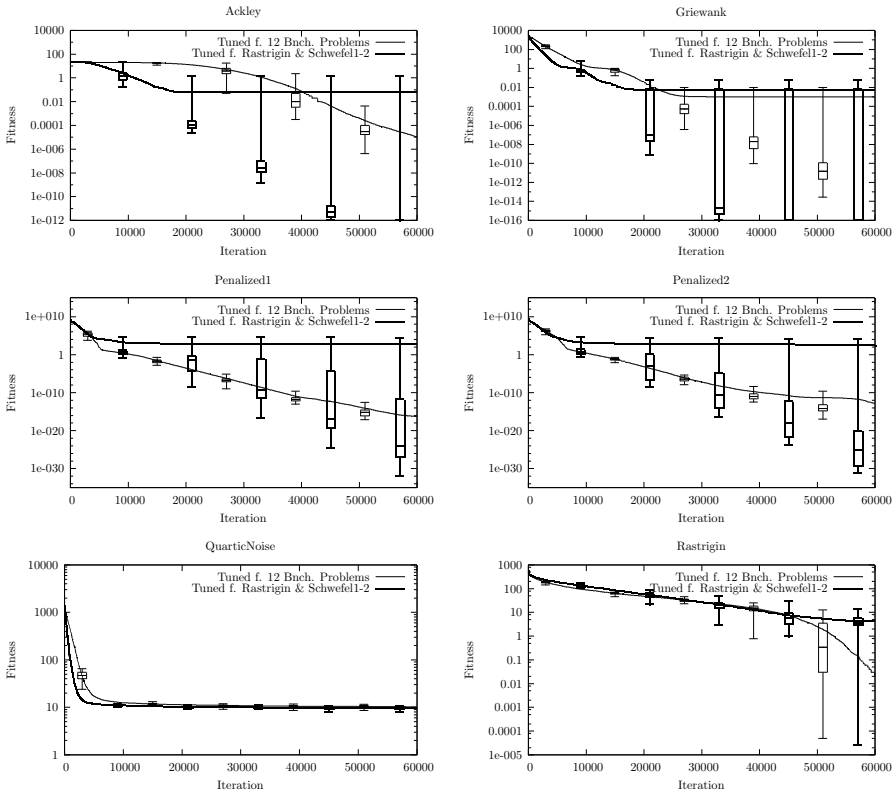

Figure 4.21: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from table 4.10, which were meta-optimized for respectively all **12 benchmark problems** and **Rastrigin & Schwefel1-2** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

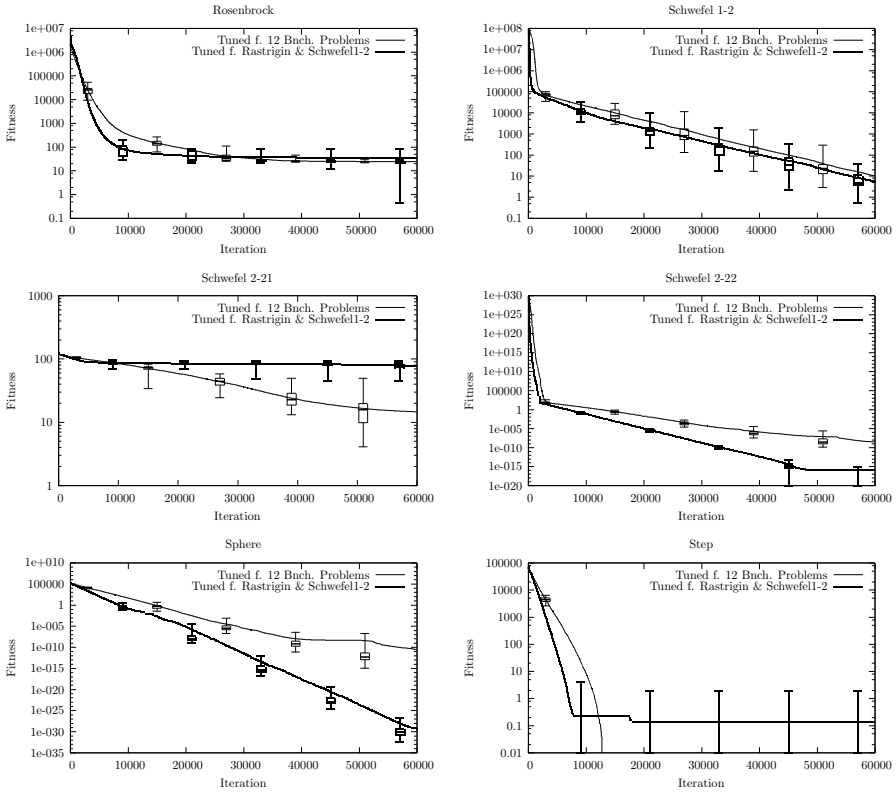

Figure 4.22: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from table 4.10, which were meta-optimized for respectively all **12 benchmark problems** and **Rastrigin & Schwefel1-2** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

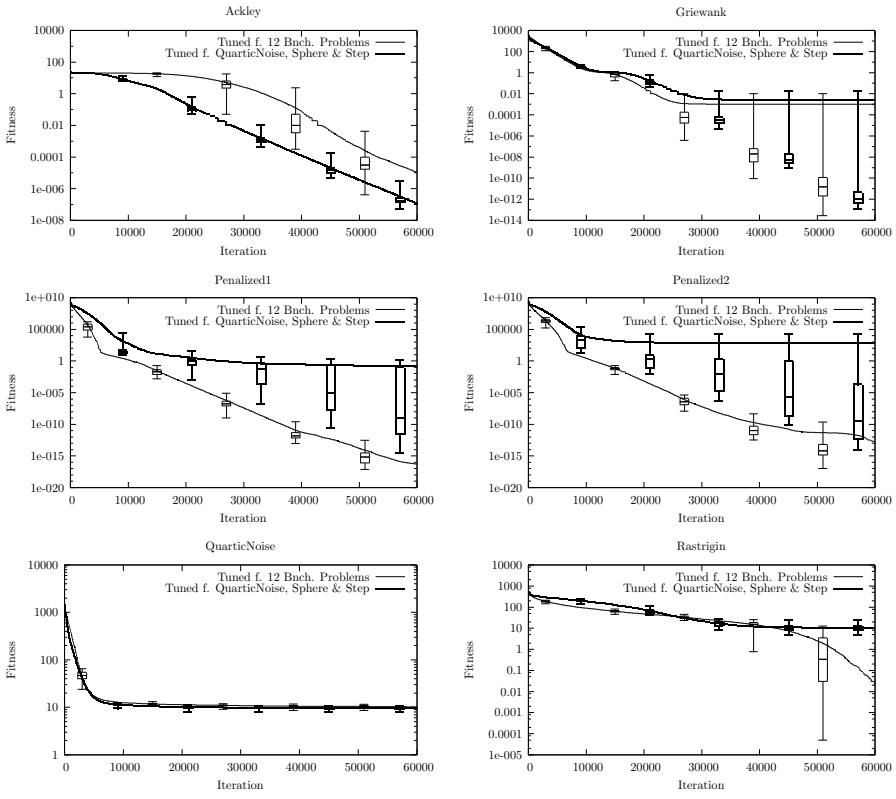

Figure 4.23: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from table 4.10, which were meta-optimized for respectively all **12 benchmark problems** and **QuarticNoise, Sphere & Step** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

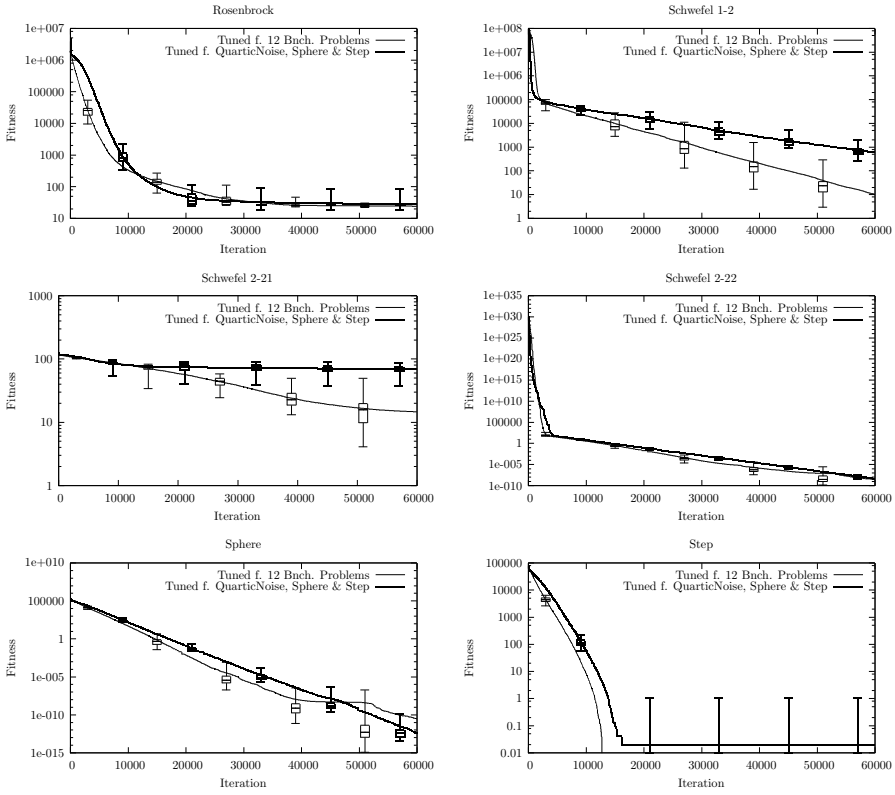

Figure 4.24: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from table 4.10, which were meta-optimized for respectively all **12 benchmark problems** and **QuarticNoise, Sphere & Step** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

	Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
All Benchmark Problems	Ackley	1.1e-5	2.74e-5	7.86e-9	6.34e-7	1.92e-6	6.15e-6	1.77e-4
	Griewank	1.04e-3	2.88e-3	1.11e-16	1.29e-14	9.4e-14	6.3e-13	0.01
	Penalized1	5e-17	2.17e-16	5.89e-21	5.55e-19	3.01e-18	1.23e-17	1.45e-15
	Penalized2	1.67e-13	1.15e-12	3.97e-20	7.78e-18	3.01e-17	2.08e-15	8.24e-12
	QuarticNoise	10.2	0.56	8.65	9.86	10.28	10.54	11.5
	Rastrigin	0.02	0.12	1.15e-8	1.73e-5	2.47e-4	5.41e-3	0.88
	Rosenbrock	24.26	1.07	21.89	23.78	24.28	24.83	27.13
	Schwefel1-2	10.38	13.26	0.63	3.2	5.96	12.45	77.18
	Schwefel2-21	14.66	10.56	1.2	6.37	13.5	18.71	49.51
	Schwefel2-22	3.2e-9	1.16e-8	2.05e-12	2.59e-11	1.34e-10	8.26e-10	7.98e-8
	Sphere	3.13e-11	2.19e-10	4.08e-18	4.61e-16	2.47e-15	2.43e-14	1.56e-9
	Step	0	0	0	0	0	0	0
Rosenbrock & Sphere	Ackley	2.49	5.73	1.36e-12	9.67e-9	2.12e-6	0.04	19.72
	Griewank	1.33e-3	3.46e-3	0	0	0	0	0.01
	Penalized1	0.01	0.07	1.57e-32	1.57e-32	3.31e-32	9.06e-32	0.52
	Penalized2	4.39e-4	2.15e-3	1.35e-32	1.47e-32	2.89e-32	8.38e-32	0.01
	QuarticNoise	10.74	2.89	8.92	9.9	10.29	10.74	30.27
	Rastrigin	13.68	9.32	1.99	6.96	10.94	18.9	45.77
	Rosenbrock	21.32	14.64	13.45	16.95	17.67	18.74	80.87
	Schwefel1-2	9965	2779	5007	7726	9623	11944	18303
	Schwefel2-21	28.36	32.22	0.14	2.07	5.73	75	75
	Schwefel2-22	0.25	1.75	0	0	0	0	12.5
	Sphere	1.7e-31	3.23e-31	1.82e-33	2.21e-32	5.01e-32	1.58e-31	1.61e-30
	Step	0	0	0	0	0	0	0
Rastrigin & Schwefel1-2	Ackley	0.06	0.26	3.11e-15	6.66e-15	6.66e-15	6.66e-15	1.34
	Griewank	5.92e-3	0.02	0	0	0	7.4e-3	0.07
	Penalized1	628.97	4402	1.57e-32	1.51e-29	2.03e-26	2.5e-14	31444
	Penalized2	264.58	1641	1.35e-32	7.27e-32	6.72e-28	1.28e-22	11714
	QuarticNoise	9.63	0.53	7.94	9.36	9.65	9.9	10.89
	Rastrigin	4.27	2.78	6.08e-7	2.98	2.99	5.97	12.43
	Rosenbrock	33.21	21	0.31	22.7	25.58	27.91	81.45
	Schwefel1-2	5.56	5.41	0.4	2.03	3.1	7.17	26.59
	Schwefel2-21	78.47	11.1	44.8	73.34	82.64	85.75	92.66
	Schwefel2-22	1.15e-16	2.32e-16	0	0	0	0	8.88e-16
	Sphere	5.16e-30	3.22e-29	2.91e-35	2.31e-33	1.54e-32	6.77e-32	2.3e-28
	Step	0.14	0.4	0	0	0	0	2
QuarticNoise, Sphere & Step	Ackley	1.14e-7	2.03e-7	1.88e-8	4.57e-8	6.18e-8	9.84e-8	1.39e-6
	Griewank	2.86e-3	6.75e-3	1.21e-14	4.57e-14	1.2e-13	7.64e-13	0.02
	Penalized1	0.14	0.36	3.1e-16	3.87e-13	1.42e-10	0.1	1.67
	Penalized2	620.64	2754	7.38e-16	4.22e-14	2.68e-11	1.95e-5	17459
	QuarticNoise	9.5	0.53	7.89	9.34	9.55	9.94	10.37
	Rastrigin	10.92	4	4.97	7.96	9.95	12.93	24.87
	Rosenbrock	29.06	13.58	17.75	25.27	26.07	26.42	82.94
	Schwefel1-2	604.81	310.13	215.16	399.98	518.83	718.01	1650
	Schwefel2-21	69.13	10.54	37.78	64.87	70.41	74.77	87.11
	Schwefel2-22	4.36e-9	1.89e-9	1.28e-9	2.87e-9	4.26e-9	5.16e-9	9.53e-9
	Sphere	4.3e-13	2.29e-12	4.07e-15	1.42e-14	3.8e-14	1.28e-13	1.64e-11
	Step	0.02	0.14	0	0	0	0	1

Table 4.12: Optimization end results for **JDE/rand/1/bin** using the behavioural parameters from table 4.10 which were meta-optimized for various combinations of benchmark problems using **60,000** fitness evaluations. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 60,000.

Problems	NP	CR	F
All Benchmark Problems	186	0.8493	0.4818
Ackley	19	0.0130	1.2935
Rastrigin	42	0.0082	0.9417

Table 4.13: Behavioural parameters for **DE/simple** that are meta-optimized for various combinations of benchmark problems in 30 dimensions each and optimization run-lengths of **60,000** iterations. Optimization results are found in figure 4.25.

Problems	NP	CR			F		
All Benchmark Problems	32	CR_{init}	=	0.0947	F_{init}	=	0.6068
		CR_l	=	0.9166	F_l	=	0.3462
		CR_u	=	0.0834	F_u	=	1.1388
		τ_{CR}	=	0.0180	τ_F	=	0.0561
Rosenbrock	19	CR_{init}	=	0.3607	F_{init}	=	1.1531
		CR_l	=	0.3110	F_l	=	0.4339
		CR_u	=	0.6890	F_u	=	1.3545
		tau_{CR}	=	0.0618	tau_F	=	0.1696
Schwefel1-2	28	CR_{init}	=	0.2309	F_{init}	=	0.9733
		CR_l	=	0.6364	F_l	=	0.2937
		CR_u	=	0.3636	F_u	=	0.7191
		tau_{CR}	=	0.0835	tau_F	=	0.9093

Table 4.14: Behavioural parameters for **JDE/rand/1/bin** that are meta-optimized for various combinations of benchmark problems in 30 dimensions each and optimization run-lengths of **60,000** iterations. Optimization results are found in figure 4.26.

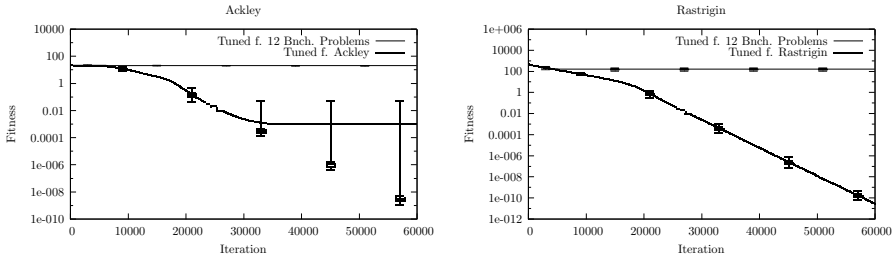

Figure 4.25: Comparison of optimization progress for **DE/simple** using the behavioural parameters from table 4.13, which were meta-optimized for respectively all **12 benchmark problems** and individually for the **Ackley** and **Rastrigin** problems using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

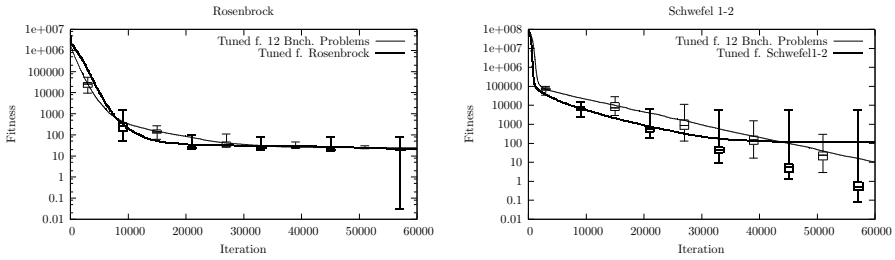

Figure 4.26: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from table 4.14, which were meta-optimized for respectively all **12 benchmark problems** and individually for the **Rosenbrock** and **Schwefel1-2** problems using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

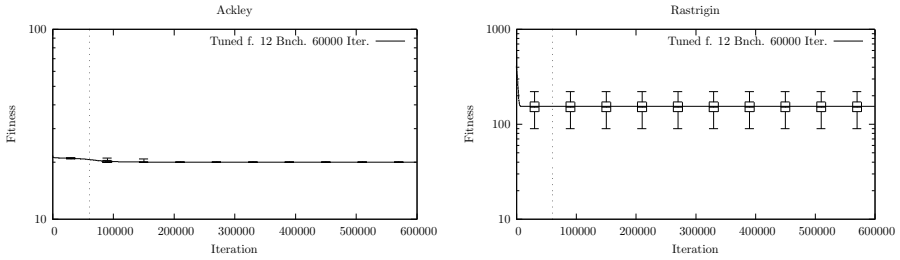

Figure 4.27: Optimization progress for **DE/simple** using the behavioural parameters from table 4.9, which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization. Dotted line shows 60,000 fitness evaluations.

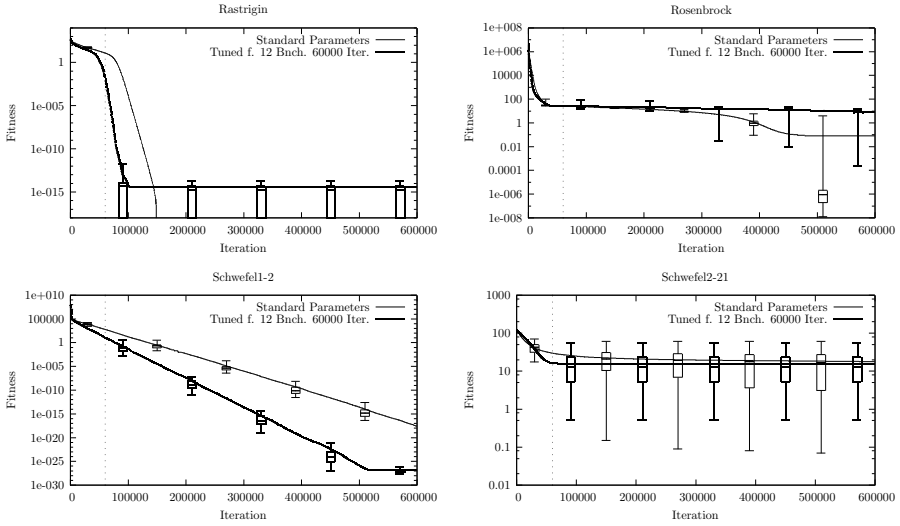

Figure 4.28: Comparison of optimization progress for **JDE/rand/1/bin** using the behavioural parameters from tables 4.2 and 4.3 which are respectively **standard** in the literature, and meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization. Dotted line shows 60,000 fitness evaluations.

DE/simple	$NP = 120$	$CR = 0.4852$	$F = 0.6413$
JDE/rand/1/bin	$NP = 82$	$CR_{init} = 0.0478$ $CR_l = 0.4655$ $CR_u = 0.5345$ $\tau_{CR} = 0.0729$	$F_{init} = 1.5127$ $F_l = 0.4136$ $F_u = 0.1835$ $\tau_F = 0.8375$

Table 4.15: Behavioural parameters for DE/simple and JDE/rand/1/bin that are meta-optimized for **Ackley, Rastrigin, Rosenbrock & Schwefel1-2** in 30 dimensions each and optimization run-lengths of **600,000** iterations. Optimization results are found in figures 4.29 and 4.30.

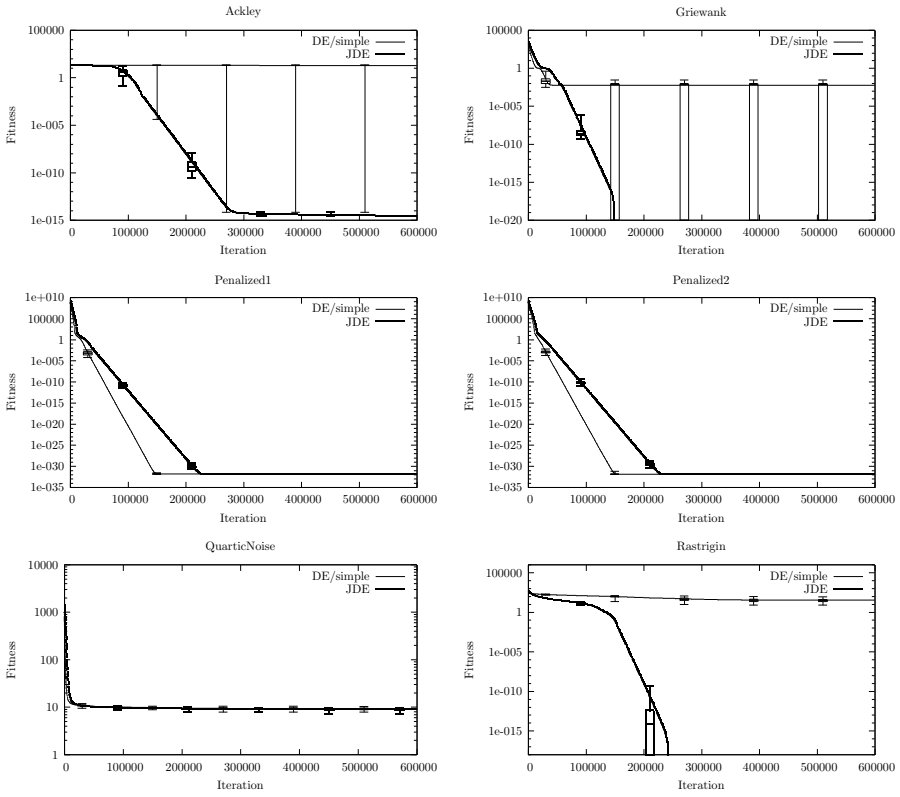

Figure 4.29: Comparison of optimization progress for DE/simple and JDE/rand/1/bin using the behavioural parameters from table 4.15, which were meta-optimized for **Ackley, Rastrigin, Rosenbrock & Schwefel1-2** using **600,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

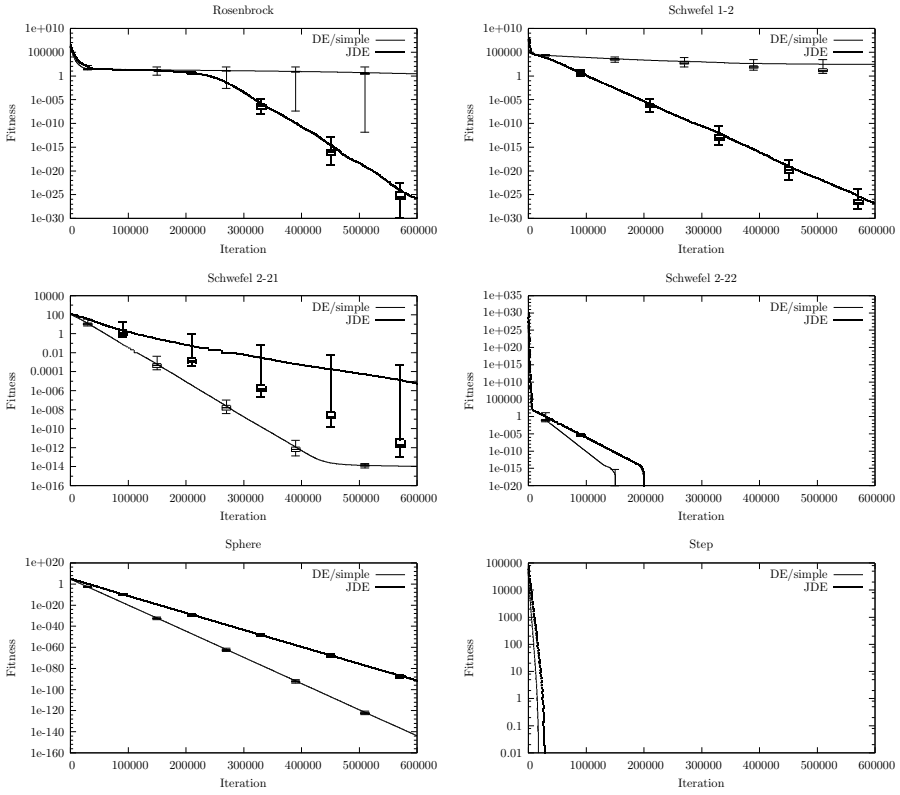

Figure 4.30: Comparison of optimization progress for DE/simple and JDE/rand/1/bin using the behavioural parameters from table 4.15, which were meta-optimized for **Ackley, Rastrigin, Rosenbrock & Schwefel1-2** using **600,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

Weights in Meta-Optimization				DE Parameters		
Ackley	Rastrigin	Rosenbrock	Schwefel1-2	NP	CR	F
1	1	1	1	191	0.8448	0.5100
10000	1	1	1	20	0.1139	0.8742
1	10000	1	1	25	0.0399	0.9704
1	1	10000	1	72	0.7722	0.4594
1	1	1	10000	195	0.9618	0.5027
10000	1000	1	1	21	0.0403	0.8817
1	1	100	100	179	0.8473	0.3325

Table 4.16: Behavioural parameters for **DE/simple** that are meta-optimized for the **Ackley, Rastrigin, Rosenbrock and Schwefel1-2** problems in 30 dimensions each and optimization run-lengths of **60,000** iterations. Various weights are used for the problems. Optimization results are found in table 4.17.

Weights	Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
1	Ackley	20.76	0.28	19.94	20.75	20.83	20.91	20.97
1	Rastrigin	156.16	27.74	104.47	137.3	154.69	176.59	217.81
1	Rosenbrock	0.95	1.58	1.21e-7	0.03	0.11	0.93	4.33
1	Schwefel1-2	120.33	550.88	8.52e-4	0.01	0.05	0.92	2813
10000	Ackley	1.05e-6	5.25e-6	2.98e-13	1.39e-11	1.42e-10	3.87e-9	3.66e-5
1	Rastrigin	5.87	4.02	0	2.98	4.97	6.96	16.91
1	Rosenbrock	26.75	8.6	18.22	24.48	25.41	26.61	81.2
1	Schwefel1-2	12566	2737	6837	10572	12213	14212	19747
1	Ackley	5.01e-7	1.26e-6	3.57e-9	2.72e-8	7.08e-8	4.14e-7	8.07e-6
10000	Rastrigin	0.16	0.61	0	0	0	0	3.98
1	Rosenbrock	29.5	13.15	18.34	24.52	26.43	28.32	80.68
1	Schwefel1-2	16829	3204	8046	15209	16795	19012	24115
1	Ackley	19.95	0.01	19.9	19.94	19.95	19.95	19.96
1	Rastrigin	160.12	26.69	106.46	139.78	155.94	174.09	245.19
10000	Rosenbrock	0.88	1.65	8.06e-13	9.15e-8	7.9e-7	1.21e-3	3.99
1	Schwefel1-2	1721	2928	3.42e-6	2.83e-4	3.84e-3	2813	13438
1	Ackley	19.95	0.03	19.89	19.94	19.95	19.96	20.13
1	Rastrigin	162.47	32.94	92.53	135.8	169.6	188.02	219.77
1	Rosenbrock	1.51	1.94	4.62e-14	1.46e-8	3.31e-7	3.99	3.99
10000	Schwefel1-2	4.54e-7	2.89e-6	4.73e-11	1.12e-9	4.28e-9	1.51e-8	2.07e-5
10000	Ackley	0.82	3.25	4.18e-12	1.9e-11	8.38e-11	2.51e-10	14.8
1000	Rastrigin	2.61	2.38	0	0.99	1.99	3.98	8.95
1	Rosenbrock	28.04	12.92	13.62	23.95	25.45	27.37	80.41
1	Schwefel1-2	11485	2423	6599	9436	11286	13280	18363
1	Ackley	19.95	9.67e-3	19.91	19.95	19.95	19.96	19.97
1	Rastrigin	168.25	22.09	113.43	154.22	167.88	183.07	234.78
100	Rosenbrock	1.04	1.75	4.92e-12	2.08e-8	2.19e-6	3.99	3.99
100	Schwefel1-2	1.1e-3	4.22e-3	6.9e-8	3.8e-6	2.22e-5	1.32e-4	0.03

Table 4.17: Optimization end results for **DE/simple** using the behavioural parameters from table 4.16, which were meta-optimized for the **Ackley, Rastrigin, Rosenbrock and Schwefel1-2** problems using **various weights** and **60,000** fitness evaluations. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 60,000.

	NP	CR	F	Meta-Fitness
DE/simple	186	0.8493	0.4818	10051
	175	0.9369	0.5417	10674
	181	0.8471	0.4074	10769
	181	0.9019	0.4425	10989
	162	0.8773	0.4087	11125
	197	0.9300	0.3580	11480
	188	0.9519	0.4047	12159
	180	0.8601	0.2635	17204
	136	0.6604	0.3701	19667
	117	0.8183	0.3661	20652
DE/rand/1/bin	19	0.1220	0.4983	112647
	19	0.1318	0.4928	114646
	24	0.2121	0.3991	117681
	28	0.3334	0.4077	129805
	31	0.3362	0.3852	129987
	37	0.3184	0.3609	132937
	46	0.4944	0.3233	182752
	79	0.2825	0.2654	183267
	80	0.2969	0.2884	188145
	79	0.3115	0.2867	190367

	NP	CR	F_{mid}	F_{range}	Meta-Fitness
DE/rand/1/bin/Dither	102	0.9637	0.7876	0.7292	10176
	88	0.9332	0.1475	1.4495	10467
	103	0.9624	0.7677	0.7092	11577
	33	0.8646	0.6680	0.2828	12428
	34	0.8590	0.6522	0.2534	13325
	110	0.9630	0.2937	1.7954	15334
	105	0.9680	0.2888	1.7478	15942
	108	0.9621	0.3525	1.8483	17914
	29	0.8375	0.6243	0.1952	18478
	110	0.9632	0.2960	1.8697	19364
DE/rand/1/bin/Jitter	58	0.9048	0.3989	0.3426	13277
	52	0.9002	0.4521	0.3646	16036
	47	0.9131	0.3776	0.4628	16475
	37	0.8867	0.3855	0.5872	17647
	49	0.8937	0.2662	0.6046	23459
	44	0.9402	0.3186	0.6775	27985
	42	0.9156	0.2208	0.7113	35914
	31	0.9216	0.2167	0.8566	51107
	25	0.9705	0.4120	0.8326	79824
	17	0.0683	0.4891	0.2944	106213

	NP	CR_{init}	CR_l	CR_u	τ_{CR}	F_{init}	F_l	F_u	τ_F	Meta-Fitness
JDE/rand/1/bin	32	0.0947	0.9166	0.7104	0.0180	0.6068	0.3462	1.1388	0.0561	3136
	31	0.1050	0.9301	0.6974	0.0169	0.6126	0.3479	1.1630	0.0637	3311
	34	0.1199	0.9230	0.7135	0.0175	0.6501	0.3406	1.1257	0.0492	3447
	36	0.1099	0.9062	0.7075	0.0221	0.6426	0.3117	1.1578	0.0340	3461
	40	0.1233	0.8859	0.6921	0.0321	0.6332	0.3389	1.1106	0.0340	4129
	37	0.0323	0.3588	0.9576	0.0500	0.3360	0.1461	1.1636	0.8474	4184
	35	0.0285	0.3406	0.9700	0.0345	0.3018	0.1139	1.1875	0.8320	4320
	34	0.0171	0.3348	0.9978	0.0441	0.3223	0.0954	1.2067	0.8563	4478
	44	0.1374	0.9041	0.6766	0.0403	0.6471	0.3144	1.0924	0.0181	4986
	46	0.1636	0.9007	0.6536	0.0264	0.6341	0.2708	1.1406	0.0444	5043

Table 4.18: Best 10 sets of behavioural parameters for DE variants that are meta-optimized for all **12 benchmark problems** in 30 dimensions each and optimization run-lengths of **60,000** iterations.

Number of Problems	Optimization Iterations	Time Usage	
		DE/simple	JDE
2	60,000	41 min	2 h 54 min
4	6,000	8 min	25 min
4	600,000	24 h 22 min	40 h 38 min
12	6,000	22 min	54 min
12	60,000	4 h 40 min	13 h 50 min

Table 4.19: Time usage for meta-optimizing the behavioural parameters of **DE/simple** and **JDE/rand/1/bin** with various numbers of benchmark problems and optimization iterations being used.

Chapter 5

Particle Swarm Optimization

5.1 Introduction

New PSO variants are continually being introduced in an attempt to improve optimization performance. Most of these variants increase the complexity of the original PSO method. To appreciate the popularity of that research approach just consider the journal volume in which the paper derived from this chapter was published [123], which also contains numerous papers with more complex PSO variants, e.g. [124] [125] [126] [127] [128] [129] [130] [131]. There are certain trends in that research, one is to make a hybrid optimization method using PSO combined with one or more other optimizers, see e.g. [76] [125] [127] [130], another trend is to try and alleviate optimization stagnation by reversing or perturbing the movement of the PSO particles, see e.g. [132] [133] [134] [124], and then there are also attempts at trying to adapt the behavioural parameters of PSO during optimization, see e.g. [135].

That the behavioural parameters play an important role in the performance of PSO has been acknowledged since its inception and several studies of the parameters have been made, see e.g. Shi and Eberhart [22] [23] and Carlisle and Dozier [24]. To properly guide selection of behavioural parameters attempts have also been made by van den Bergh [27], Clerc and Kennedy [29] and Trelea [28] at mathematically analyzing the PSO behaviour by considering how the parameters influence diversity of the particles, with the assumption apparently being that diversity in turn influences optimization efficacy. To facilitate these analyses a number of other assumptions were also made:

- The particles' points of attraction $\vec{p}$ and $\vec{g}$ in Eq.(2.3) remain constant so that improved positions found in the search-space have no influence on the swarm's further optimization progress.
- This also means that just a single particle is considered and not an entire

swarm of particles as there is no inter-particle communication through $\vec{g}$ which now remains unchanged.

- The stochastic variables r_p and r_g in Eq.(2.3) are also eliminated by using their expectancies instead.

But do these features not seem to be what makes PSO work at all?

This chapter studies a basic PSO and derived simplifications. Of the many studies that have been published only very few have been on simplifying the PSO method. Kennedy [51] studied simplifications to the PSO method but unfortunately did not have the necessary tools to make a rigorous comparison with the basic PSO, something that will be done here. A more recent study is due to Bratton and Blackwell [52] who made performance plots akin to those in chapter 3, but unfortunately only considered positive behavioural parameters where it is found in this chapter that PSO often performs best with negative parameters. That a more rigorous study can easily be made here is due to the use of meta-optimization for automatically tuning the PSO parameters. This was also done by Meissner et al. [32] but their study was limited in terms of the accuracy and quality of the results obtained, as their approach lacked the proper choice of the overlaid meta-optimizer, it lacked time-saving features, and it did not have the ability to meta-optimize the PSO parameters with regard to their performance on multiple optimization problems.

5.2 Simplifications

This chapter will focus on the basic PSO from chapter 2 and three simplifications, they are:

- PSO is the basic variant from chapter 2 whose velocity update formula is rewritten here for convenience:

$$\vec{v} \leftarrow \omega\vec{v} + \phi_p r_p(\vec{p} - \vec{x}) + \phi_g r_g(\vec{g} - \vec{x}) \tag{5.1}$$

- PSO-VG is the variant that only has velocity (V) and attraction to the swarm's best known position (G). The velocity update formula is:

$$\vec{v} \leftarrow \omega\vec{v} + \phi_g r_g(\vec{g} - \vec{x}) \tag{5.2}$$

  Note that PSO-VG is also referred to as the MOL method in [123] and the "social only" PSO by Kennedy [51].

- PSO-PG is the variant that has no recurrent velocity but only attraction to the particle's own best known position (P) and the swarm's best known position (G). The position update formula is:

$$\vec{x} \leftarrow \vec{x} + \phi_p r_p(\vec{p} - \vec{x}) + \phi_g r_g(\vec{g} - \vec{x})$$

- PSO-G is the variant that only has attraction to the swarm's best known position (G). The position update formula is:

$$\vec{x} \leftarrow \vec{x} + \phi_g r_g(\vec{g} - \vec{x})$$

5.3 Experimental Results

5.3.1 Overall Meta-Optimized Performance

The purpose of this experiment is to establish how well the PSO variants perform when their behavioural parameters have been tuned for all 12 benchmark problems simultaneously using 60,000 iterations for each optimization run on each of these problems. The boundaries for the behavioural parameters in meta-optimization are:

$$S \in \{1, \cdots, 200\},\ \omega \in [-2, 2],\ \phi_p \in [-4, 4],\ \phi_g \in [-4, 4]$$

These boundaries were chosen amply wide surrounding the standard parameters, see below, and the boundaries also allow for negative parameters where studies in the literature are usually only made for positive parameters, see e.g. [51] [24] [52], so this may yield unusual choices of behavioural parameters. The meta-optimization settings are identical to those in chapter 4, namely that 5 meta-optimization runs are performed, each consisting of a number of iterations equal to 20 times the number of behavioural parameters being tuned, that is, for basic PSO which has 4 parameters it means that 80 iterations are being performed in each meta-optimization run. The parameters thus found are shown in table 5.1. These can be compared to the standard parameters from section 2.7 which are reprinted here for convenience:

$$S = 50,\ \omega = 0.729,\ \phi_1 = \phi_2 = 1.49445 \tag{5.3}$$

The most notable differences are that the swarm size S has more than doubled and the so-called inertia weight ω has become negative which is contrary to established guidelines in the literature, see e.g. [87] [24]. All PSO variants have a large swarm-size S and a fairly large ϕ_g which is the attraction towards the swarm's best known position in the search-space. Note that the parameters of PSO-VG have S and ϕ_g close to the boundary values thus suggesting the boundaries should have been wider, but meta-optimization experiments with new boundaries $S \in \{1, \cdots, 400\}$ and $\phi_g \in [-4, 8]$ did not yield better parameters.

Figures 5.1 and 5.2 compare the optimization progress of basic PSO when using the standard and meta-optimized parameters from whence it can be seen that performance is greatly improved overall. There is a tiny advantage to the standard parameters in the first few thousand optimization iterations on the Rosenbrock and Schwefel2-22 problems but these are so negligible that they may just be due to stochastic variation.

Figures 5.3 and 5.4 show the mean fitness progress for the four PSO variants from where it can be seen that the PSO and PSO-VG are clearly the best performing and that PSO-PG and PSO-G cannot even optimize the simple Sphere problem. Since these two PSO variants employ no recurrent velocity for their optimizing particles it would suggest that the velocity is an essential part of what makes PSO work. Figures 5.5 and 5.6 show the progress of PSO and

PSO-VG in more detail where PSO-VG can be seen to have a clear advantage. The end results are shown in table 5.2 from which it can be seen that the PSO and especially the PSO-VG variant often comes close to the optimal fitness values of zero on these problems. Both variants seem to have some difficulty optimizing the Rastrigin and Schwefel1-2 problems and the basic PSO also has difficulty with the Rosenbrock problem. Note that the QuarticNoise problem is in fact optimized well and a fitness of, say, 10 is just the noise that has been added.

The first conclusion to be made is that meta-optimization greatly improves the performance of PSO in this optimization scenario. The second conclusion is that PSO can apparently be simplified to the PSO-VG variant with overall performance improvement.

5.3.2 Generalization Ability

The purpose of this experiment is to test the generalization ability of PSO and PSO-VG. Practitioners would prefer not to have to tune the behavioural parameters of an optimizer for each new problem encountered, so the ability of an optimizer to perform well on problems for which its behavioural parameters were not specifically tuned is important. In order to test this, three different sets of benchmark problems will be used in meta-optimization and the performance on the remainder of the benchmark problems will then be studied. The sets of benchmark problems are the same as for DE tested in section 4.4.5, and for similar reasons:

- Rosenbrock & Sphere, one hard and one easy problem.
- Rastrigin & Schwefel1-2, because PSO and PSO-VG had some difficulty optimizing these problems.
- QuarticNoise, Sphere & Step, because they are all unimodal problems with QuarticNoise having noise added and Step being a discontinuous version of Sphere. Parameters tuned for these simple problems may not generalize well to harder problems.

The behavioural parameters that were meta-optimized with regard to these three sets of benchmark problems are shown in table 5.3. For both PSO variants the swarm-size S has decreased significantly compared to when the parameters were tuned for all 12 benchmark problems. The parameters ω and ϕ_g show clear tendencies. For ω the values are still negative although with more magnitude than before, $\omega \simeq -0.35$ for both PSO variants as opposed to $\omega \simeq -0.16$ for PSO and $\omega \simeq -0.27$ for PSO-VG before. The parameter ϕ_g is also somewhat similar for the two PSO variants, although it seems that when ϕ_p is large and positive for the basic PSO variant then ϕ_g must be smaller. For PSO-VG the good values seem to be around $\phi_g \simeq 3.5$. The intrinsic meaning of these choices of parameters is unknown.

Figures 5.7 and 5.8 compare the optimization progress of PSO and PSO-VG when using the parameters tuned for Rosenbrock & Sphere. Note from

table 5.3 the PSO parameter $\phi_p = -0.0075$ which is so close to zero that we would expect it to perform on par with PSO-VG which is really just PSO with $\phi_p = 0$. Indeed, the progress plots show that performance is very similar, with a slight advantage of PSO-VG on e.g. the Ackley and Rosenbrock problems, and a slight disadvantage on e.g. the Schwefel1-2 and 2-22 problems. Considering, however, the proximity to the optimal fitness values of zero the performance is deemed comparable, as expected.

Figures 5.9 and 5.10 compare the optimization progress of using the parameters tuned for Rastrigin & Schwefel1-2. The performance is overall very similar for PSO and PSO-VG. The most notable difference is on the Ackley problem where PSO-VG has worse mean progress but the quartiles reveal that it actually discovers better solutions than PSO on occasion.

Figures 5.11 and 5.12 compare the optimization progress of using the parameters tuned for QuarticNoise, Sphere and Step. Here, PSO-VG seems to have an advantage overall, perhaps with a slight exception on the Rastrigin problem on which PSO-VG performs marginally worse.

Table 5.4 shows the optimization results of using PSO with these behavioural parameters. Interestingly, the parameters tuned for Rosenbrock & Sphere and for Rastrigin & Schwefel1-2 also seem to improve the end results on many of the other problems. This is also partially true for the parameters tuned for QuarticNoise, Sphere & Step, but on e.g. the Schwefel1-2 problem performance is worsened compared to when the parameters were tuned for all 12 benchmark problems.

Table 5.5 shows the end results of optimization using PSO-VG. Here the tendencies are not so clear. For instance, tuning the parameters for Rosenbrock & Sphere leads to slight improvement on these two problems where performance was already good, but the mean results on e.g. Schwefel1-2 and 2-22 are now worse, although the quartiles do reveal that better solutions are in fact being found and worse solutions are found only occasionally. Performance on some of the other problems is improved, see e.g. the Rastrigin problem. When the parameters were tuned for the Rastrigin & Schwefel1-2 problems the quartiles are improved for the Schwefel1-2 problem but worsened for the Rastrigin problem compared to when the parameters were tuned for all 12 benchmark problems. Performance on the remaining problems seems comparable overall. When the parameters were tuned for the QuarticNoise, Sphere & Step problems performance seems to be comparable overall to when the parameters were tuned for all 12 benchmark problems.

The conclusion is that both PSO and PSO-VG appear to generalize well to problems for which their behavioural parameters were not specifically tuned. PSO-VG seemed to have a small advantage when its parameters were tuned for the simple problems QuarticNoise, Sphere & Step, but otherwise PSO and PSO-VG seemed to be comparable in terms of their generalization ability.

5.3.3 Specialization Ability

The purpose of this experiment is to uncover how well PSO and PSO-VG can be made to perform by tuning their behavioural parameters for individual optimization problems. This is useful in practise if a large number of closely related problems must be optimized but the common choice of behavioural parameters do not yield satisfactory results.

It was found in the experiments above that PSO consistently had difficulties optimizing the Rastrigin and Schwefel1-2 problems, but also the Rosenbrock and sometimes the Ackley problems. Table 5.6 shows the parameters of PSO tuned for these problems individually which are very different from the parameters tuned for all 12 benchmark problems. It would be pure speculation to say why these parameters should perform better on these specific problems. Two interesting observations can be made, though. First, that the behavioural parameters are all negative when tuned to work well on the Rastrigin problem. Second, when the parameters are tuned for the Rosenbrock problem the swarm size S only has two particles and all the behavioural parameters are now positive. Figure 5.13 shows the optimization progress of using these parameters compared to when the behavioural parameters were tuned for all 12 benchmark problems. It can be seen that performance is improved on these individual benchmark problems, but performance is still irregular and satisfactory solutions are not always found to these problems, see especially the Schwefel1-2 problem where the mean and median fitness are well above the optimal fitness value of zero.

For PSO-VG the experiments above have shown difficulty optimizing the Rastrigin and Schwefel1-2 problems. Table 5.6 shows the behavioural parameters for PSO-VG tuned for these two problems individually, which appear somewhat related to the parameters that were tuned for all 12 benchmark problems. This is interesting because the basic PSO variant had very different parameters and it therefore indicates that introducing the ϕ_p parameter greatly alters and perhaps distorts the meta-fitness landscape, while it is seemingly more smooth for the PSO-VG variant that does not have the ϕ_p parameter. Figure 5.14 shows the optimization progress of using these specialized parameters compared to the parameters that were tuned for all 12 benchmark problems. On the Rastrigin problem the quartiles show that performance is improved for most but not all of the optimization runs. On the Schwefel1-2 problem performance is actually worsened when the parameters are tuned specifically for that one problem. This is interesting but the cause is unknown. A guess would be that it is due to the general performance irregularity of PSO-VG on the Schwefel1-2 problem which somehow diverts the meta-optimizer from locating the region of good behavioural parameters, and this irregularity is perhaps masked when tuning the behavioural parameters for multiple problems.

The conclusion is that performance of PSO and PSO-VG could be improved on some problems when the behavioural parameters were tuned specifically for one optimization problem at a time, although both these PSO variants still had difficulty with the Schwefel1-2 problem on which the specialized parameters of PSO-VG actually caused worse performance.

5.3.4 Long Optimization Runs

The purpose of this experiment is to uncover the performance capabilities when PSO and PSO-VG are allowed 600,000 iterations in optimization instead of 60,000 as before. Table 5.8 shows the end results of using the hand-tuned parameters from section 2.7 which are clearly still not capable of finding near-optimal fitness values (close to zero), even though ten times as many optimization iterations are now being performed.

Figures 5.15 and 5.16 show the optimization progress of using the behavioural parameters for PSO and PSO-VG that were meta-optimized for all 12 benchmark problems when using 60,000 optimization iterations, marked by the dotted line in these plots. It can be seen that PSO-VG has a small advantage over PSO on these problems overall and PSO-VG is capable of finding solutions with near-optimal fitness values for all problems while PSO is having some difficulty on especially the Rastrigin problem.

Tuning the behavioural parameters for all 12 benchmark problems using 600,000 optimization iterations instead yields the parameters in table 5.7. The PSO parameters are quite different from before but the PSO-VG parameters are somewhat similar. Figures 5.17 and 5.18 show the optimization progress of using these parameters where it can be seen that PSO-VG progresses slightly faster and better on most of the problems, although the results are so close to the optimal fitness values of zero that performance should be deemed comparable. Table 5.10 shows the end results and PSO has occasional difficulty in optimizing the Rastrigin problem while PSO-VG has occasional difficulty in optimizing the Schwefel2-22 problem.

Tuning the behavioural parameters for only the Rastrigin and Schwefel1-2 problems using 600,000 optimization iterations yields the parameters in table 5.7. The PSO parameters seem to be somewhat related to when they were tuned for all 12 benchmark problems using 600,000 optimization iterations, while the PSO-VG parameters show similar tendencies to before but of different magnitudes. Figures 5.19 and 5.20 show the optimization progress of using these parameters. On both of the problems for which the behavioural parameters were tuned PSO-VG performs slightly worse than PSO, although the proximity to the optimal fitness values of zero suggests they should perhaps be deemed comparable. On the problems for which the behavioural parameters were not tuned there does seem to be a general advantage of PSO, although the proximity to optimal fitness values again suggests that the performance should be deemed comparable. Indeed, table 5.10 shows that near-optimal fitness values are frequently found for all problems by both PSO and PSO-VG, but PSO-VG occasionally has difficulties with especially Schwefel2-21 and 2-22. This is interesting because the behavioural parameters of PSO-VG were previously tuned to perform and generalize better than those of PSO. The cause of this deficiency is unknown, but a guess would be that tuning the PSO-VG parameters for Rastrigin and Schwefel1-2 causes a similar sort of confusion for the meta-optimizer as was observed when tuning the parameters for just the Schwefel1-2 problem in section 5.3.3 above.

The conclusion is that both PSO and PSO-VG can be tuned to perform well in these longer optimization runs, although some care must be taken in selecting the problems for which the behavioural parameters are tuned, so as to make the performance generalize well to other problems.

5.3.5 Deterministic Variants

The purpose of this experiment is to uncover the performance capabilities of deterministic PSO and PSO-VG variants, which are identical to their original stochastic versions except for fixing the stochastic variables, that is, $r_p = r_g = 1$ in Eqs.(5.1) and (5.2) for the velocity updates. Recall that deterministic PSO variants were the subject of the mathematical analyses in [27] [29] [28], so this experiment may confirm or disconfirm the usefulness of those analyses.

Table 5.9 shows the behavioural parameters of the deterministic PSO variants that were discovered through meta-optimization. Comparing these to the parameters for the stochastic PSO variants in table 5.1 the most striking differences are that the inertia weight is no longer negative and that the weight ϕ_g is significantly lower than before. The optimization results of using these parameters are shown in table 5.11. Clearly, the deterministic PSO variants perform poorly compared to the stochastic variants, especially PSO-VG has greatly diminished optimization capabilities when it is made deterministic.

The conclusion is that PSO variants perform significantly better when their movement in the search-space is stochastic rather than deterministic. Theoretical analyses in the literature that have assumed determinism of PSO would therefore seem to be misleading in practise.

5.3.6 Parameter Study

This section is a more detailed study of the behavioural parameters from the meta-optimization experiments above. Table 5.12 lists the ten best sets of parameters found in some of these experiments. Recall that smaller meta-fitness values indicate better optimization performance. Observe that good choices of ω apparently have to be negative for PSO and PSO-VG both, and shorter optimization runs seem to need an ω parameter of smaller magnitude. For PSO good choices of the ϕ_g parameter seem to be roughly $\phi_g \in [2; 3.5]$ although negative values are also sometimes found to work well. The ϕ_p parameter is more erratic and it is hard to see a clear tendency for what choices of ϕ_p yield good performance. For PSO-VG good choices of the ϕ_g parameter are fairly consistently found to be close to the boundary value of $\phi_g \simeq 4$ for the optimization runs having 60,000 iterations, while smaller values, say, $\phi_g \in [2; 3]$ are found to work well for the longer optimization runs. For PSO a good choice of swarm-size seems to be roughly $S \in \{70, \cdots, 100\}$ and for PSO-VG it should perhaps be somewhat higher, say, $S \in \{70, \cdots, 200\}$ depending on the problem configuration.

Figure 5.21 shows meta-fitness landscapes of using PSO-VG with a fixed swarm-size $S = 100$, which seems to be a good compromise from the above. The

landscapes are made for the Rastrigin, Rosenbrock and Schwefel1-2 problems individually. There is obvious correlation between the parameters that work well for these problems, namely $\omega \in [-0.5; -0.25]$ and $\phi_g \in [3; 4]$, although positive values of ω also work well on the Rosenbrock problem but not on the other two problems. This region of good parameters is consistent with the meta-optimized parameters from table 5.12. Figure 5.22 shows the meta-fitness landscapes for combinations of benchmark problems with a similar region of good performing parameters.

Observe from the plots in figures 5.21 and 5.22 as well as the behavioural parameters listed in table 5.12 that good choices of the ϕ_g parameter are so close to its boundary value of 4. This was already noted in section 5.3.1 where an additional meta-optimization experiment was conducted with wider boundaries for the S and ϕ_g parameters but did not yield better results. Perhaps that conclusion was too hasty. To investigate if wider boundaries are really necessary figure 5.23 shows the meta-fitness landscapes where the upper boundary for ϕ_g has been expanded to 16 instead of 4. Also note that the swarm-size is now fixed to $S = 200$ instead of $S = 100$, this was done to get a slightly different view of the meta-fitness landscape, from which it can be seen that the swarm-size does not appear to greatly influence the performance of PSO-VG. As can be seen from figure 5.23 good choices of ϕ_g seem to be in the range $\phi_g \in [2.5; 5]$. This is perhaps more evident from the plots in figure 5.24 and would suggest that all the meta-optimization experiments above should indeed have been conducted with slightly wider boundaries. However, the best behavioural parameter for the Rastrigin & Schwefel1-2 problems when traversing this grid was indeed found to be $\phi_g = 3.6923$, and $\phi_g = 3.1795$ for the grid measuring the performance on all 12 benchmark problems. Both these ϕ_g values are below the upper boundary of 4 as used in the meta-optimization experiments, thus suggesting the boundaries were in fact wide enough. Nevertheless, the proximity to the boundary might explain why there were occasional difficulties in locating good parameters, because of the small size of the region and that the surrounding *valley* was cut off. See for example table 5.12 where meta-optimization had difficulty finding good behavioral parameters for the Rastrigin & Schwefel1-2 problems when using 600,000 optimization iterations. Although a good parameter choice was indeed found to be $\phi_g \simeq 2$ which is well below the boundary of 4, there were only two good sets of parameters found which indicates difficulty in parameter tuning. Redoing that meta-optimization experiment with the boundaries $S \in \{1, \cdots, 300\}$ and $\phi_g \in [-4; 6]$ yields the parameters in table 5.13 which now all have good performance as indicated by the low meta-fitness measure. Using the best of these parameters for PSO-VG gives the optimization progress shown in figures 5.25 and 5.26 that are compared to PSO, which did not seem to need re-tuning with wider parameter boundaries as the parameters in table 5.12 are well within the boundaries. The performance of PSO and PSO-VG is now mostly comparable, perhaps with a slight advantage to PSO-VG as shown by the quartiles, although the results are close to the optimal fitness value of zero for PSO and PSO-VG both.

The overall conclusion is that PSO-VG seems to have more predictable and

consistently good behavioural parameters than PSO, although the boundaries for the parameters in meta-optimization must be sufficiently wide or the region of good parameters can be difficult to locate. On the other hand, the boundaries should also not be made too wide because the good region will then again become difficult to find. Chapter 6 has suggestions on how to overcome the dilemma of selecting boundaries for behavioural parameters.

5.3.7 Time Usage

The time usage for some of the previous meta-optimization experiments have been listed in table 5.14. The experimental settings were similar to those for DE as described in section 4.4.10. As can be seen, PSO-VG with its 3 behavioural parameters is somewhat faster to tune than PSO with 4 parameters. That the time usage does not scale exactly in terms of the number of behavioural parameters being tuned, the number of benchmark problems and the number of optimization iterations, is due to the different effect that Preemptive Fitness Evaluation has on different optimizers in different scenarios.

Now compare the time usage of meta-optimization for all 12 benchmark problems with that of traversing a 40x40 grid of the space of behavioural parameters and computing the meta-fitness value at each point, as was done for the meta-fitness landscape plotted in figure 5.22 which took more than 90 hours to compute. That was just for 2 behavioural parameters, computing a grid of that resolution for all 3 parameters of PSO-VG would take about 150 days, and for PSO with its 4 parameters it would take more than 16 years. Meta-optimization for that problem configuration took less than 3 hours for PSO-VG as shown in table 5.14, and less than 4 hours for PSO. This clearly shows the advantage of using meta-optimization instead of a more exhaustive, grid-based search for good behavioural parameters.

5.3.8 Comparison to Differential Evolution

Comparison of PSO and DE variants will now be done for a few overall performance characteristics. First is the performance when the behavioural parameters were tuned for all 12 benchmark problems using 60,000 optimization iterations. This is compared for PSO-VG and DE/Simple because they are the simplified optimizer variants and they were both found to work well in that scenario. Figures 5.27 and 5.28 show the optimization progress which is reprinted from the previous plots for easy comparison. First note that DE/Simple performs poorly on the Ackley and Rastrigin problems, which was a deficiency of DE/Simple already known from the study in chapter 4. But on most of the remaining problems, e.g. the Griewank, Penalized1 and 2, Rosenbrock, Schwefel1-2, 2-21 and 2-22 problems DE/Simple gets closer to the optimal fitness value of zero towards the end of the optimization runs than PSO-VG does.

Another DE variant will now be used in the performance comparison for 600,000 optimization iterations. For longer optimization runs JDE was found to work better than DE/Simple in chapter 4, mainly because DE/Simple could

not have its parameters tuned to work well for all benchmark problems simultaneously. The behavioural parameters for JDE were tuned for the Ackley, Rastrigin, Rosenbrock and Schwefel1-2 problems, while the parameters for PSO-VG were only tuned for the Rastrigin and Schwefel1-2 problems. This would seem to make for an unfair comparison, but considering the strong correlation of PSO-VG parameters tuned for different problems, as shown in section 5.3.6, there seems to be little to gain in tuning the PSO-VG parameters for two additional problems. Figures 5.29 and 5.30 show the optimization progress. As can be seen, PSO-VG performs better than JDE early during optimization but JDE is able to get closer to the optimal fitness values of zero later during optimization. Considering the proximity of the PSO-VG results to the optimal fitness values of zero, it is however questionable if JDE is a real improvement as the PSO-VG results are likely good enough for practical purposes.

Comparing the time usage of meta-optimizing the behavioural parameters of PSO-VG and JDE, as shown in tables 5.14 and 4.19, it is evident that it takes considerable longer to tune the 9 parameters of JDE than it takes to tune the 3 parameters of PSO-VG. In particular, consider meta-optimization for all 12 benchmark problems and 60,000 optimization iterations, where JDE takes almost 14 hours to tune while PSO-VG takes less than 3 hours. Also note that the time usage of tuning the parameters of DE/Simple is almost twice that of PSO-VG, in spite of them both having 3 parameters to be tuned. The reason for this is that Preemptive Fitness Evaluation yields different time savings for different optimization methods.

The conclusion is that the simple PSO-VG variant progresses somewhat faster in optimization, but the DE variants can get closer to the optimal fitness values for these benchmark problems. Though, the proximity of the PSO-VG results to optimal fitness values may be good enough for practical purposes.

5.4 Summary

From the experiments in this chapter a number of conclusions can be made. First, tuning the behavioural parameters of PSO variants greatly improved optimization performance. Second, the PSO parameters that worked best were contrary to guidelines in the literature. Third, the simplified PSO-VG variant generally performed on par with the basic PSO from which it was derived, and sometimes even had improved performance. The behavioural parameters of PSO-VG were also easier to tune although their boundaries should be chosen properly. Fourth, stochastic PSO algorithms perform significantly better than deterministic ones. Fifth, PSO-VG had slightly faster optimization progress than the DE variants, but did not get quite as close to the optimal fitness values, although the results of PSO-VG may be satisfactory in practise.

Variant	S	ω	ϕ_p	ϕ_g
PSO	134	-0.1618	1.8903	2.1225
PSO-VG	198	-0.2723	-	3.8283
PSO-PG	113	-	-3.3743	1.6645
PSO-G	133	-	-	3.7858

Table 5.1: Behavioural parameters for PSO variants meta-optimized for all **12 benchmark problems** in 30 dimensions each and optimization run-lengths of **60,000** iterations. Optimization results are found in table 5.2 and figures 5.1-5.6.

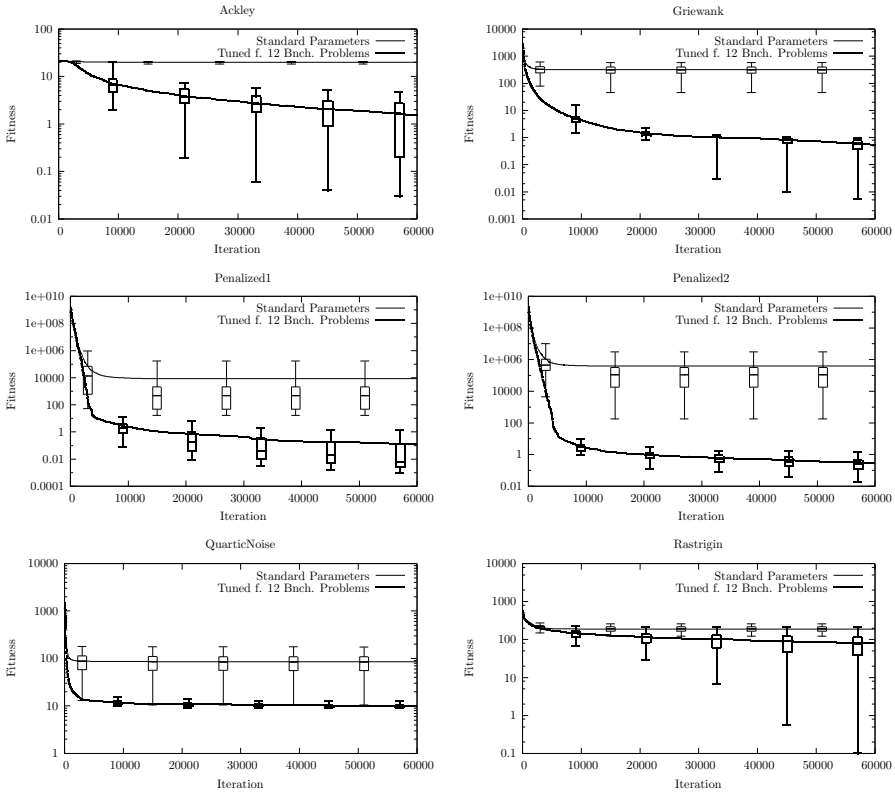

Figure 5.1: Comparison of optimization progress for **basic PSO** using the behavioural parameters from Eq.(5.3) which are standard in the literature and the parameters from table 4.3 which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

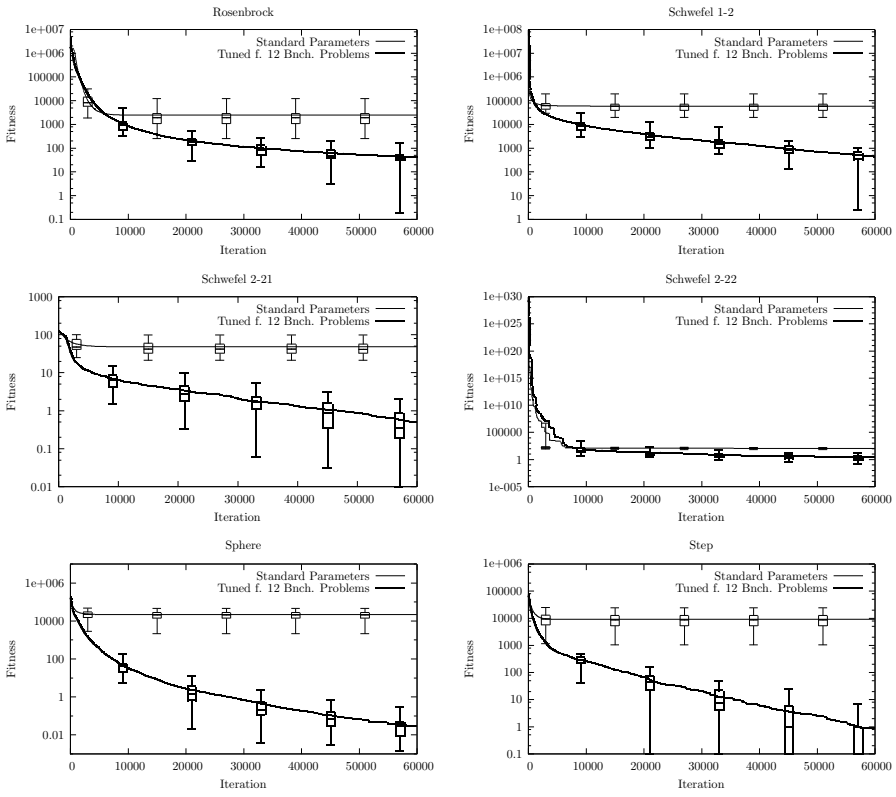

Figure 5.2: Comparison of optimization progress for **basic PSO** using the behavioural parameters from Eq.(5.3) which are standard in the literature and the parameters from table 4.3 which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

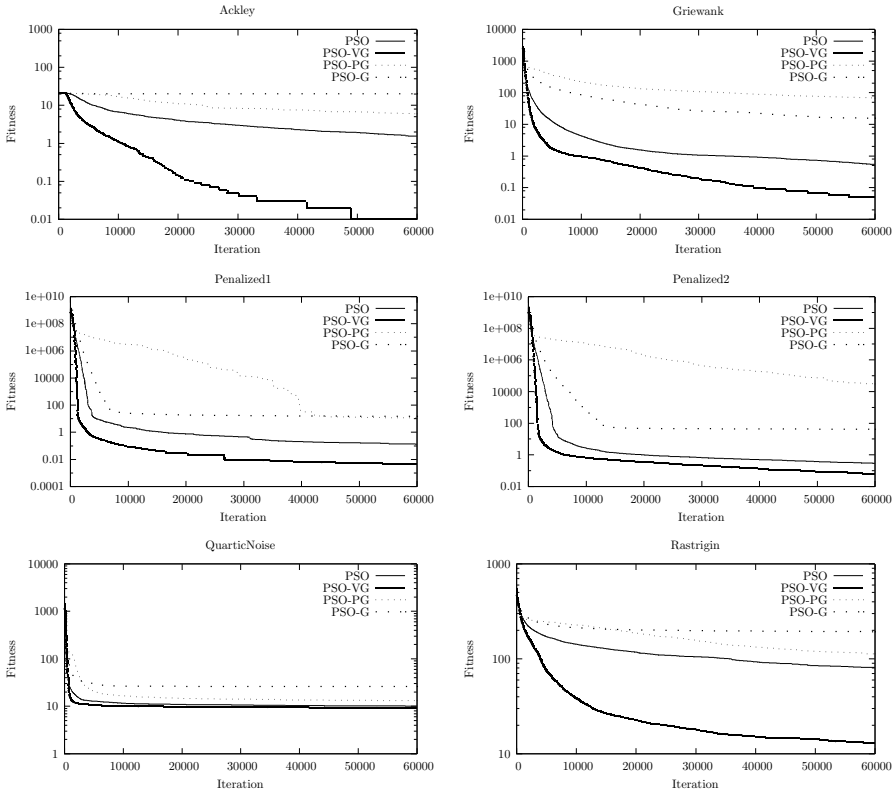

Figure 5.3: Comparison of optimization progress for PSO variants using the behavioural parameters from table 5.1 which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs.

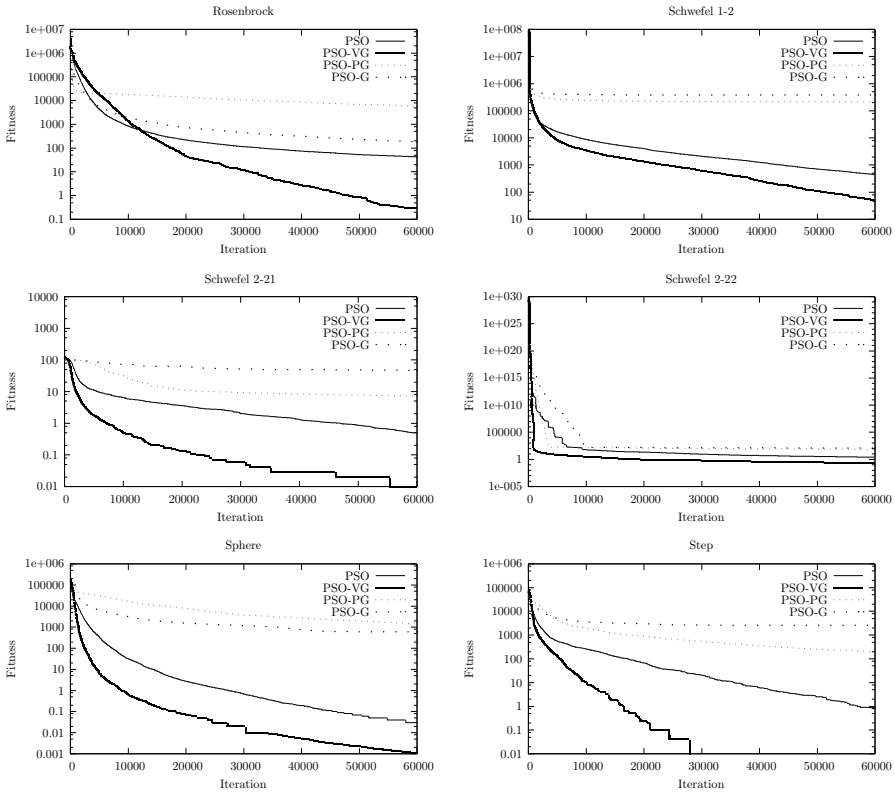

Figure 5.4: Comparison of optimization progress for PSO variants using the behavioural parameters from table 5.1 which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs.

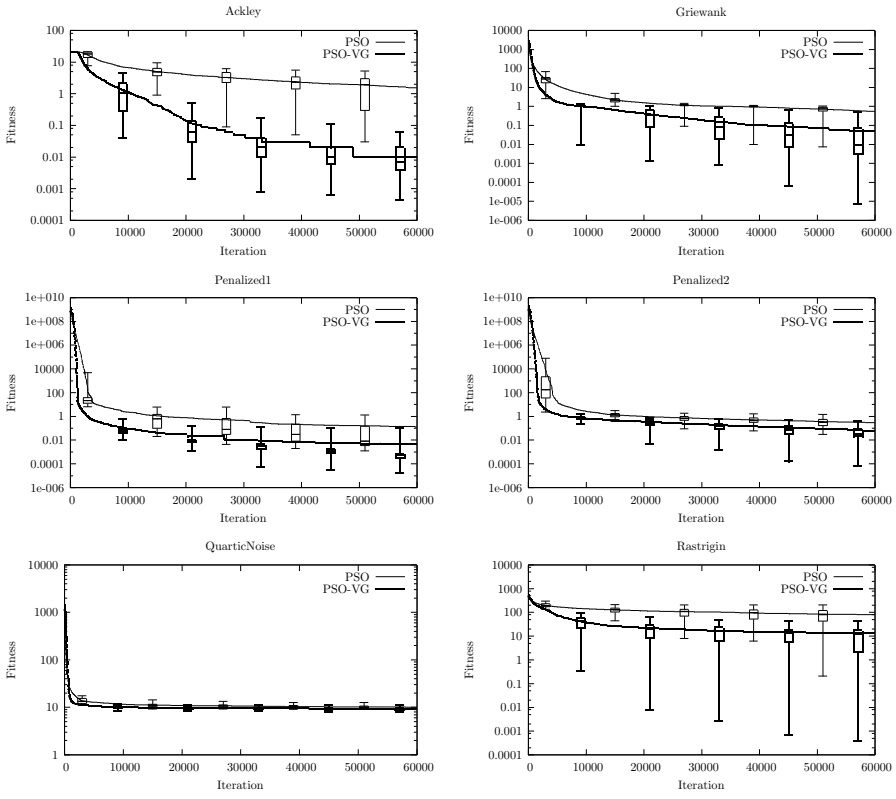

Figure 5.5: Comparison of optimization progress for **PSO** and **PSO-VG** using the behavioural parameters from table 5.1 which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

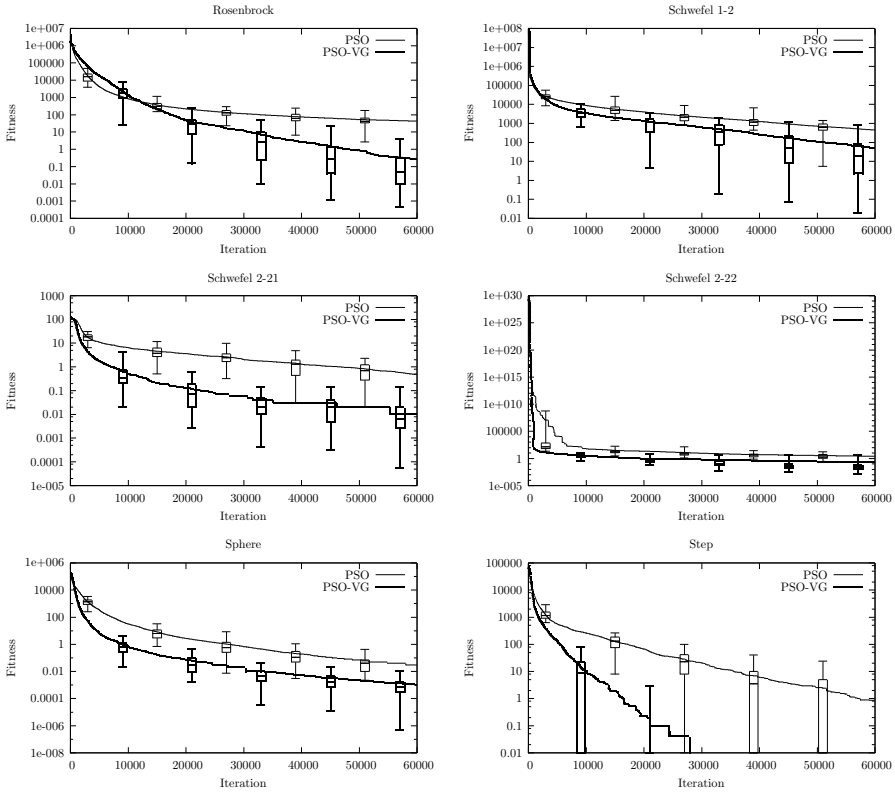

Figure 5.6: Comparison of optimization progress for **PSO** and **PSO-VG** using the behavioural parameters from table 5.1 which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

	Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
PSO	Ackley	1.54	1.33	0.03	0.19	1.35	2.66	4.42
	Griewank	0.54	0.26	5.13e-3	0.36	0.58	0.74	0.98
	Penalized1	0.13	0.28	8.19e-4	2.03e-3	5.57e-3	0.13	1.29
	Penalized2	0.29	0.25	0.01	0.12	0.23	0.41	1.35
	QuarticNoise	10.12	0.75	8.83	9.6	10.06	10.55	12.64
	Rastrigin	81	56.64	0.09	38.23	74.61	118.5	207.86
	Rosenbrock	42.77	29.21	0.14	31.44	35.81	47.67	152.52
	Schwefel1-2	449.19	231.1	1.9	277.51	426.35	593.53	982.47
	Schwefel2-21	0.5	0.47	0.01	0.18	0.27	0.75	1.93
	Schwefel2-22	2.65	3.03	0.16	0.65	1.14	5.19	15.5
	Sphere	0.03	0.04	1.23e-3	6.81e-3	0.02	0.04	0.21
	Step	0.88	1.86	0	0	0	1	7
PSO-VG	Ackley	0.01	0.01	4.03e-4	3.8e-3	6.98e-3	0.01	0.05
	Griewank	0.05	0.08	5.09e-6	1.92e-3	8.11e-3	0.07	0.5
	Penalized1	4.74e-3	0.02	1.43e-5	2.15e-4	4.12e-4	6.86e-4	0.11
	Penalized2	0.06	0.08	6.38e-5	0.01	0.03	0.06	0.45
	QuarticNoise	9.27	0.56	8.11	8.89	9.25	9.64	10.93
	Rastrigin	13	11.54	3.5e-4	2.18	12.5	18.04	45.01
	Rosenbrock	0.29	0.67	4.3e-4	8.13e-3	0.04	0.2	3.63
	Schwefel1-2	46.9	97.87	0.02	1.5	10.69	49.83	632.05
	Schwefel2-21	0.01	0.02	2.99e-5	2.73e-3	5.68e-3	0.02	0.15
	Schwefel2-22	0.24	0.97	5.85e-4	9.31e-3	0.03	0.06	5.02
	Sphere	1.07e-3	1.54e-3	4.56e-7	2.4e-4	6.01e-4	1.4e-3	9.93e-3
	Step	0	0	0	0	0	0	0
PSO-PG	Ackley	6.05	2.76	3.53	4.52	5.19	6.23	16.19
	Griewank	67.79	24.4	24.25	53.68	65.97	84.92	128.65
	Penalized1	11.43	7.06	1.82	5.08	10.51	15.3	28.09
	Penalized2	30679	164175	3.01	37.95	69.05	6589	1.17e+6
	QuarticNoise	13.17	1.83	9.76	11.82	12.99	14.33	18.42
	Rastrigin	112.52	66.9	11.05	47.9	98.51	178.76	234.75
	Rosenbrock	5758	2959	2147	3321	5139	7916	16853
	Schwefel1-2	212371	531057	4598	11513	18029	90739	3.14e+6
	Schwefel2-21	7.16	4.49	0.89	3.71	6.79	9.01	22.93
	Schwefel2-22	69.95	60.6	10.35	22.4	41.46	107.32	192.91
	Sphere	1412	910.47	2.31	829.54	1260	1893	5519
	Step	201.78	104.03	49	126	171.5	253	540
PSO-G	Ackley	20.06	0.1	19.96	19.98	20.03	20.14	20.4
	Griewank	15.55	44.02	0.75	1.04	1.08	1.2	192.28
	Penalized1	14.95	5.91	6.33	10.4	14.35	18.09	35.84
	Penalized2	41.56	12.26	5.95	36.49	42.54	49.74	70.57
	QuarticNoise	26.05	13.25	12.54	16.53	22.11	28.89	71.38
	Rastrigin	193.22	44.43	106.81	169.43	180.48	212.53	333.63
	Rosenbrock	182.63	83.44	63.15	120.05	164.43	223.64	429.25
	Schwefel1-2	379438	856116	9547	31923	55203	252545	5.36e+6
	Schwefel2-21	45.85	43.82	1.68	7.53	16.64	99.14	102.82
	Schwefel2-22	123.68	47.17	15.88	89.68	127.8	159.04	220.59
	Sphere	600.02	2338	1.02	3.44	5.39	8.89	10002
	Step	2573	2280	216	1086	1617	3155	11709

Table 5.2: Optimization end results for PSO variants using the behavioural parameters from table 5.1 which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 60,000.

	Problems	S	ω	ϕ_p	ϕ_g
PSO	All Benchmark Problems	134	-0.1618	1.8903	2.1225
	Rosenbrock & Sphere	84	-0.3036	-0.0075	3.9730
	Rastrigin & Schwefel1-2	82	-0.3794	-0.2389	3.5481
	QuarticNoise, Sphere & Step	50	-0.3610	0.7590	2.2897
PSO-VG	All Benchmark Problems	198	-0.2723	-	3.8283
	Rosenbrock & Sphere	42	-0.4055	-	3.1722
	Rastrigin & Schwefel1-2	47	-0.3000	-	3.5582
	QuarticNoise, Sphere & Step	83	-0.3461	-	3.2535

Table 5.3: Behavioural parameters for PSO variants that are meta-optimized for various combinations of benchmark problems in 30 dimensions each and optimization run-lengths of **60,000** iterations. Optimization results are found in figures 5.7-5.12 and tables 5.4 and 5.5.

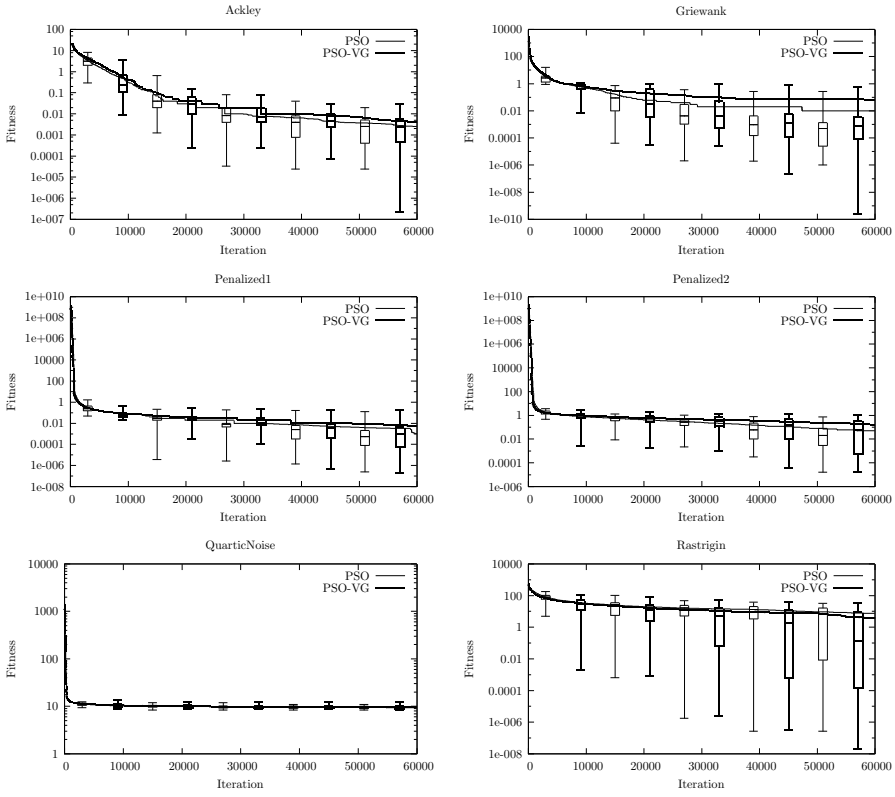

Figure 5.7: Comparison of optimization progress for PSO variants using the behavioural parameters from table 5.3 which were meta-optimized for **Rosenbrock & Sphere** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

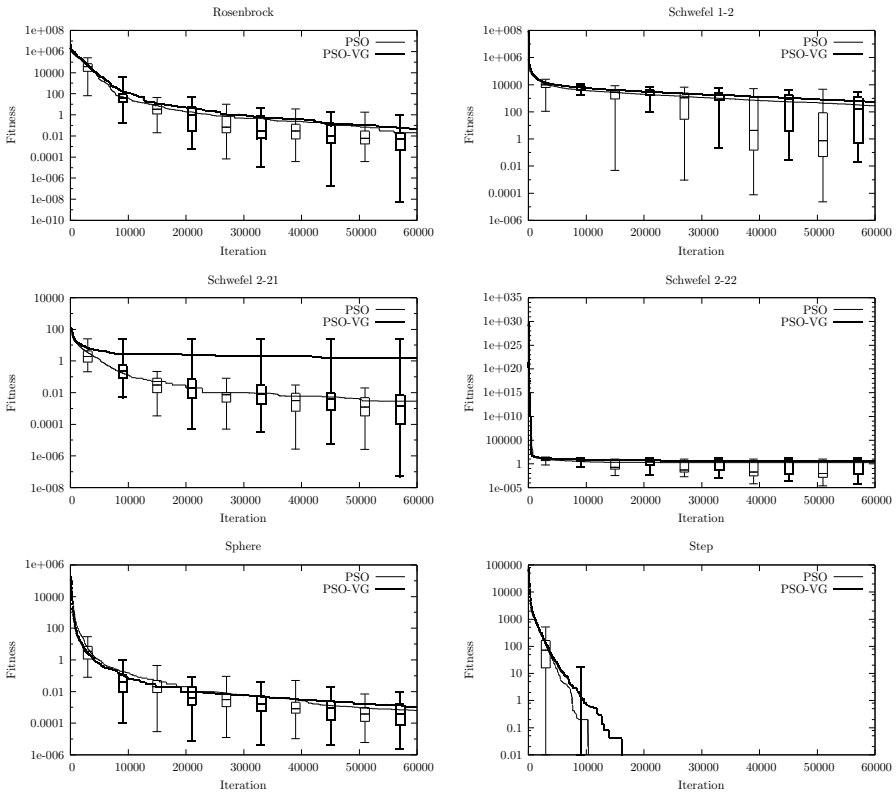

Figure 5.8: Comparison of optimization progress for PSO variants using the behavioural parameters from table 5.3 which were meta-optimized for **Rosenbrock & Sphere** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

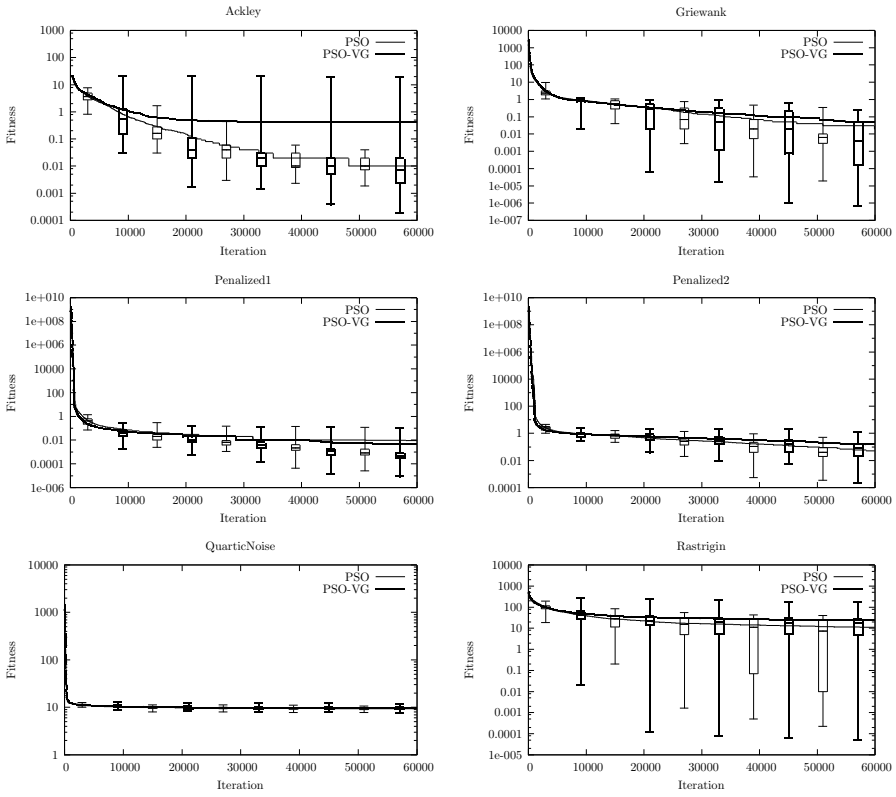

Figure 5.9: Comparison of optimization progress for PSO variants using the behavioural parameters from table 5.3 which were meta-optimized for **Rastrigin & Schwefel1-2** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

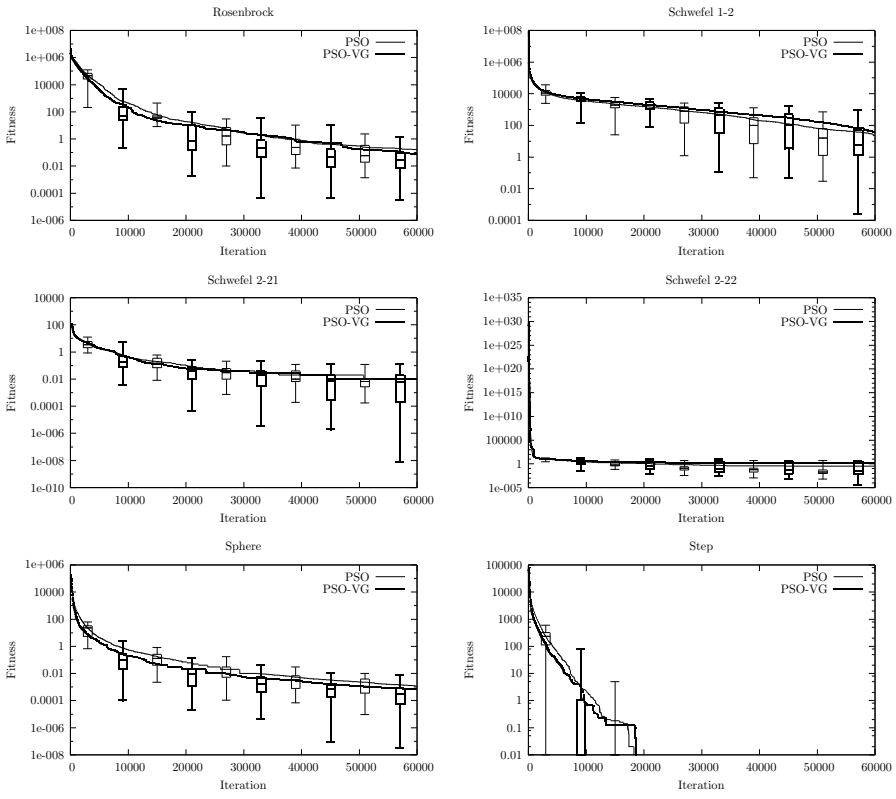

Figure 5.10: Comparison of optimization progress for PSO variants using the behavioural parameters from table 5.3 which were meta-optimized for **Rastrigin & Schwefel1-2** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

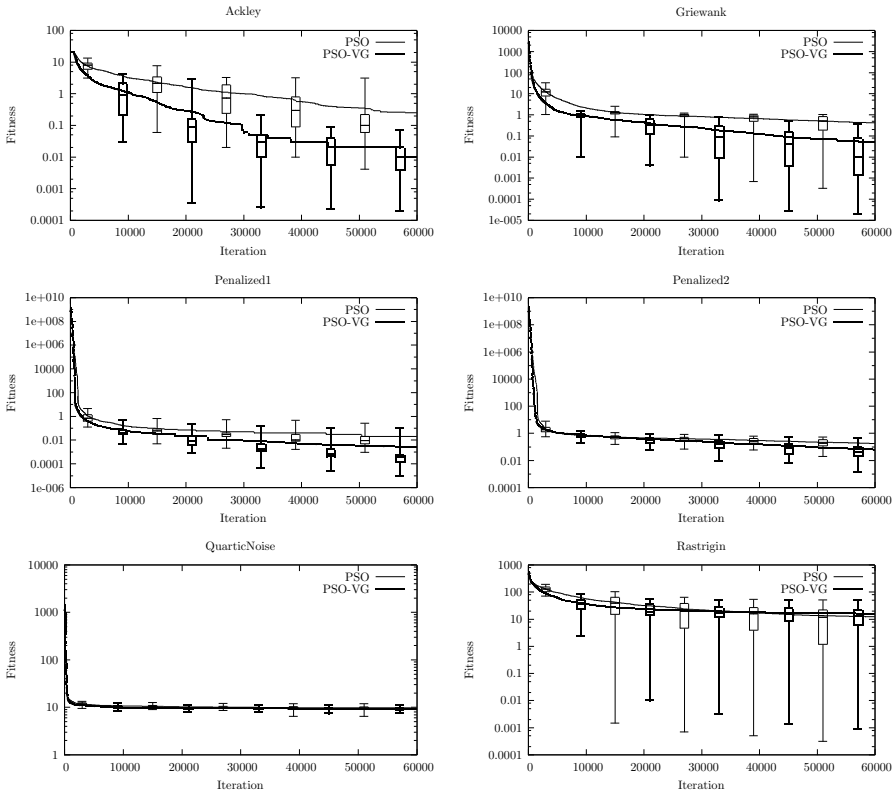

Figure 5.11: Comparison of optimization progress for PSO variants using the behavioural parameters from table 5.3 which were meta-optimized for **Quartic-Noise, Sphere & Step** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

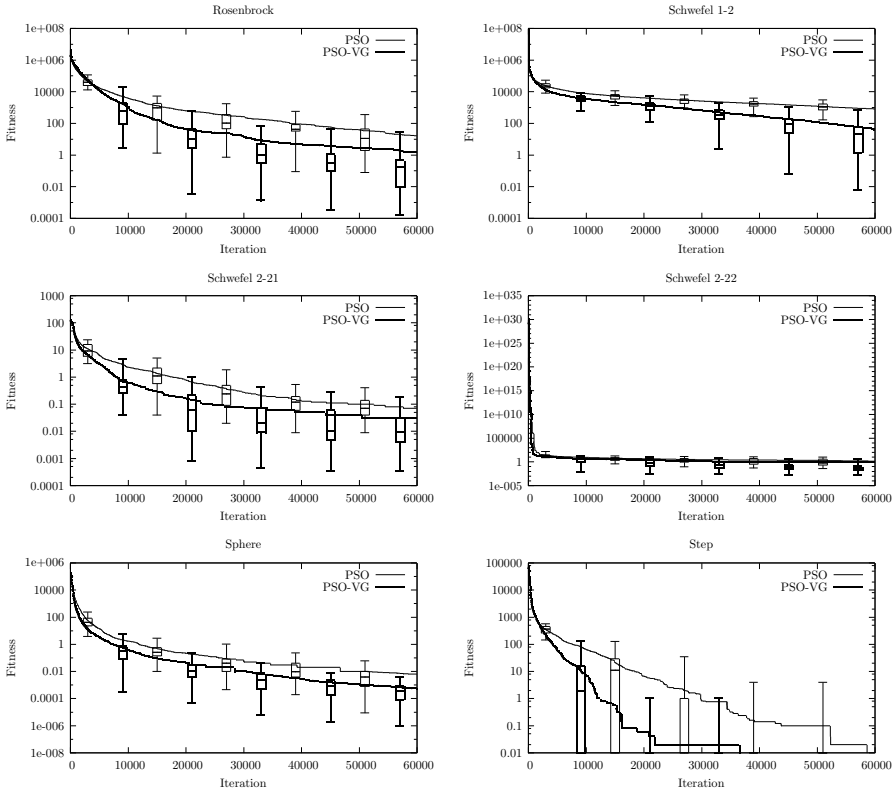

Figure 5.12: Comparison of optimization progress for PSO variants using the behavioural parameters from table 5.3 which were meta-optimized for **Quartic-Noise, Sphere & Step** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

	Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
All Benchmark Problems	Ackley	1.54	1.33	0.03	0.19	1.35	2.66	4.42
	Griewank	0.54	0.26	5.13e-3	0.36	0.58	0.74	0.98
	Penalized1	0.13	0.28	8.19e-4	2.03e-3	5.57e-3	0.13	1.29
	Penalized2	0.29	0.25	0.01	0.12	0.23	0.41	1.35
	QuarticNoise	10.12	0.75	8.83	9.6	10.06	10.55	12.64
	Rastrigin	81	56.64	0.09	38.23	74.61	118.5	207.86
	Rosenbrock	42.77	29.21	0.14	31.44	35.81	47.67	152.52
	Schwefel1-2	449.19	231.1	1.9	277.51	426.35	593.53	982.47
	Schwefel2-21	0.5	0.47	0.01	0.18	0.27	0.75	1.93
	Schwefel2-22	2.65	3.03	0.16	0.65	1.14	5.19	15.5
	Sphere	0.03	0.04	1.23e-3	6.81e-3	0.02	0.04	0.21
	Step	0.88	1.86	0	0	0	1	7
Rosenbrock & Sphere	Ackley	2.49e-3	2.78e-3	3.82e-6	3.44e-4	1.54e-3	3.94e-3	0.01
	Griewank	0.01	0.04	4.02e-10	1.89e-5	3.04e-4	9.51e-4	0.18
	Penalized1	1.03e-3	3.68e-3	2.02e-7	3.76e-5	1.8e-4	7.54e-4	0.03
	Penalized2	0.05	0.11	1.65e-5	1.07e-3	0.01	0.05	0.69
	QuarticNoise	9.38	0.63	8.33	8.91	9.3	9.88	10.9
	Rastrigin	7.56	8.89	2.72e-7	1.18e-3	3.86	14.07	31.76
	Rosenbrock	0.02	0.04	1.67e-5	2.98e-4	2.72e-3	7.16e-3	0.2
	Schwefel1-2	276.92	696.64	2.21e-5	0.02	0.16	25.15	2820
	Schwefel2-21	2.89e-3	3.92e-3	1.77e-6	1.64e-4	8.64e-4	4.8e-3	0.02
	Schwefel2-22	1.71	2.93	2.3e-5	1.27e-3	8.34e-3	5	10.02
	Sphere	6.41e-4	9.64e-4	5.54e-6	8.28e-5	3.05e-4	8.9e-4	4.17e-3
	Step	0	0	0	0	0	0	0
Rastrigin & Schwefel1-2	Ackley	0.01	8.68e-3	1.4e-3	5.13e-3	9.25e-3	0.01	0.04
	Griewank	0.02	0.06	1.59e-5	1.27e-3	4.44e-3	0.01	0.29
	Penalized1	9.33e-3	0.03	1.53e-5	2.23e-4	5.77e-4	9.77e-4	0.11
	Penalized2	0.05	0.08	1.84e-4	4.49e-3	0.03	0.05	0.46
	QuarticNoise	9.34	0.58	7.77	9.01	9.41	9.77	10.21
	Rastrigin	11.07	12.84	9.06e-5	2.84e-3	4.75	23.2	40.16
	Rosenbrock	0.17	0.32	8.39e-4	0.01	0.04	0.25	2.01
	Schwefel1-2	23.35	35.32	7.47e-3	0.49	4.7	29.19	131.21
	Schwefel2-21	0.01	0.02	7.31e-5	2.02e-3	4.75e-3	0.01	0.1
	Schwefel2-22	0.33	1.19	6.33e-4	8.74e-3	0.02	0.04	5.03
	Sphere	1.09e-3	1.33e-3	2.92e-6	2.02e-4	5.33e-4	1.63e-3	6.21e-3
	Step	0	0	0	0	0	0	0
QuarticNoise, Sphere & Step	Ackley	0.25	0.55	3.89e-3	0.03	0.07	0.16	3.05
	Griewank	0.42	0.32	2.75e-4	0.07	0.4	0.74	0.97
	Penalized1	0.02	0.04	1.4e-4	3.3e-3	5.77e-3	0.01	0.24
	Penalized2	0.18	0.13	2.71e-3	0.08	0.16	0.3	0.44
	QuarticNoise	9.7	0.77	6.46	9.35	9.63	10.18	11.81
	Rastrigin	12.34	11.94	2.68e-4	0.58	10.88	21.79	50.11
	Rosenbrock	16.62	25.56	0.03	0.84	3.45	19.38	103.9
	Schwefel1-2	822.91	470.61	123.13	466.07	757.04	1153	2260
	Schwefel2-21	0.07	0.06	4.53e-3	0.03	0.05	0.08	0.3
	Schwefel2-22	1.41	2.23	0.03	0.15	0.31	1.17	10.04
	Sphere	6.26e-3	0.01	4.62e-6	5.91e-4	2.46e-3	6.61e-3	0.06
	Step	0	0	0	0	0	0	0

Table 5.4: Optimization end results for **PSO** using the behavioural parameters from table 5.3 which were meta-optimized for various combinations of benchmark problems using **60,000** fitness evaluations. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 60,000.

	Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
All Benchmark Problems	Ackley	0.01	0.01	4.03e-4	3.8e-3	6.98e-3	0.01	0.05
	Griewank	0.05	0.08	5.09e-6	1.92e-3	8.11e-3	0.07	0.5
	Penalized1	4.74e-3	0.02	1.43e-5	2.15e-4	4.12e-4	6.86e-4	0.11
	Penalized2	0.06	0.08	6.38e-5	0.01	0.03	0.06	0.45
	QuarticNoise	9.27	0.56	8.11	8.89	9.25	9.64	10.93
	Rastrigin	13	11.54	3.5e-4	2.18	12.5	18.04	45.01
	Rosenbrock	0.29	0.67	4.3e-4	8.13e-3	0.04	0.2	3.63
	Schwefel1-2	46.9	97.87	0.02	1.5	10.69	49.83	632.05
	Schwefel2-21	0.01	0.02	2.99e-5	2.73e-3	5.68e-3	0.02	0.15
	Schwefel2-22	0.24	0.97	5.85e-4	9.31e-3	0.03	0.06	5.02
	Sphere	1.07e-3	1.54e-3	4.56e-7	2.4e-4	6.01e-4	1.4e-3	9.93e-3
	Step	0	0	0	0	0	0	0
Rosenbrock & Sphere	Ackley	3.94e-3	5.85e-3	2.28e-7	3.62e-4	2.31e-3	5.38e-3	0.03
	Griewank	0.06	0.14	2.61e-10	2.83e-5	7.12e-4	3.53e-3	0.54
	Penalized1	5.61e-3	0.02	1.91e-7	3.96e-5	3.97e-4	2.21e-3	0.17
	Penalized2	0.17	0.23	4.57e-6	5.42e-4	0.04	0.33	1.08
	QuarticNoise	9.65	0.86	8.22	9.04	9.45	10.1	12.23
	Rastrigin	3.87	5.93	1.93e-8	9.04e-5	6.3e-3	7.23	27.35
	Rosenbrock	0.05	0.13	5.68e-9	4.39e-4	5.13e-3	0.02	0.73
	Schwefel1-2	541.3	738.71	5.03e-3	0.5	102.96	1091	2810
	Schwefel2-21	1.5	5.94	5.16e-8	6.45e-5	1.33e-3	7.19e-3	25
	Schwefel2-22	4.81	4.58	3.24e-6	7.5e-3	5	10	15
	Sphere	1.06e-3	1.89e-3	2.26e-6	4.99e-5	2.84e-4	1.23e-3	9.61e-3
	Step	0	0	0	0	0	0	0
Rastrigin & Schwefel1-2	Ackley	0.41	2.8	1.66e-4	2.33e-3	5.98e-3	0.02	20.02
	Griewank	0.05	0.07	6.79e-7	7.82e-5	3.71e-3	0.04	0.24
	Penalized1	4.83e-3	0.02	6.7e-6	2.62e-4	4.13e-4	6.44e-4	0.11
	Penalized2	0.15	0.22	1.54e-4	0.02	0.06	0.14	1.25
	QuarticNoise	9.57	0.83	7.54	8.84	9.62	10.16	11.83
	Rastrigin	24.49	34.47	2.29e-6	5.04	17.98	26.07	168.2
	Rosenbrock	0.08	0.19	2.23e-5	5.14e-3	0.02	0.08	1.25
	Schwefel1-2	39.58	76.79	2.59e-4	0.41	3.62	42.36	387.69
	Schwefel2-21	0.01	0.02	6.25e-9	1.98e-4	6.22e-3	0.02	0.12
	Schwefel2-22	1.43	2.24	3e-5	5.51e-3	0.04	5	5.18
	Sphere	6.66e-4	1.09e-3	3.21e-8	4.37e-5	2.8e-4	6.55e-4	5.54e-3
	Step	0	0	0	0	0	0	0
QuarticNoise, Sphere & Step	Ackley	0.01	0.01	1.67e-4	3.58e-3	8.37e-3	0.02	0.06
	Griewank	0.05	0.07	1.98e-5	1.06e-3	0.01	0.07	0.3
	Penalized1	2.6e-3	0.01	8.41e-6	1.23e-4	2.81e-4	4.86e-4	0.11
	Penalized2	0.06	0.08	9.52e-4	0.01	0.03	0.09	0.42
	QuarticNoise	9.32	0.71	7.48	8.86	9.25	9.72	11.22
	Rastrigin	15.63	12.88	1.1e-4	4.99	13.52	22.01	52.27
	Rosenbrock	1.56	5.63	1.43e-4	0.01	0.13	0.41	29.56
	Schwefel1-2	39.8	76.19	3.38e-3	0.86	15.85	44.56	487.84
	Schwefel2-21	0.03	0.05	3.39e-4	4.05e-3	9.58e-3	0.03	0.19
	Schwefel2-22	0.76	1.74	1.72e-3	0.01	0.04	0.11	5.21
	Sphere	5.39e-4	7.43e-4	6.59e-7	6.91e-5	3.06e-4	6.74e-4	3.74e-3
	Step	0	0	0	0	0	0	0

Table 5.5: Optimization end results for **PSO-VG** using the behavioural parameters from table 5.3 which were meta-optimized for various combinations of benchmark problems using **60,000** fitness evaluations. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 60,000.

Variant	Problems	S	ω	ϕ_p	ϕ_g
PSO	All Benchmark Problems	134	-0.1618	1.8903	2.1225
	Ackley	24	-0.6421	-3.9845	0.2583
	Rastrigin	53	-1.3131	-0.7090	-0.5648
	Rosenbrock	2	0.7622	1.3619	3.4249
	Schwefel1-2	119	-0.3718	-0.2031	3.2785
PSO-VG	All Benchmark Problems	198	-0.2723	-	3.8283
	Rastrigin	114	-0.3606	-	3.8220
	Schwefel1-2	138	-0.4774	-	2.3943

Table 5.6: Behavioural parameters for PSO variants that are meta-optimized for individual benchmark problems in 30 dimensions and optimization run-lengths of **60,000** iterations. Optimization results are found in figures 5.13 and 5.14.

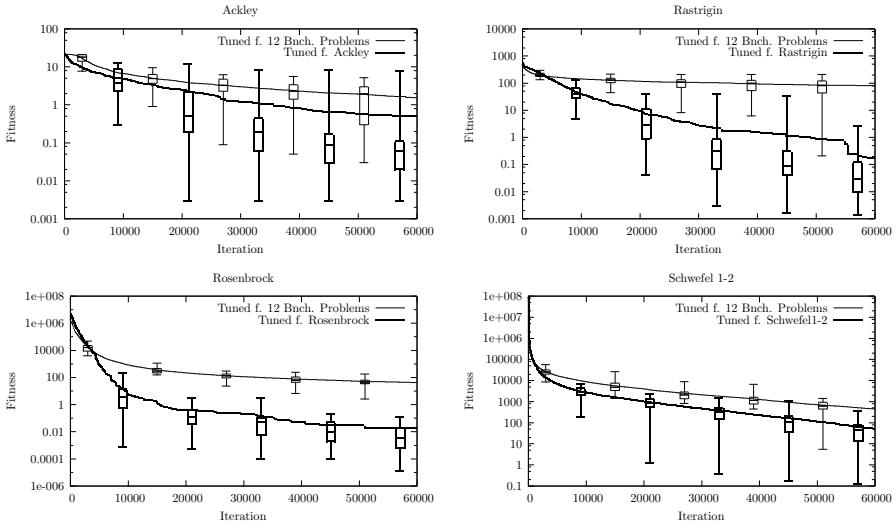

Figure 5.13: Comparison of optimization progress for **PSO** using the behavioural parameters from table 5.6 which were meta-optimized for the benchmark problems individually using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

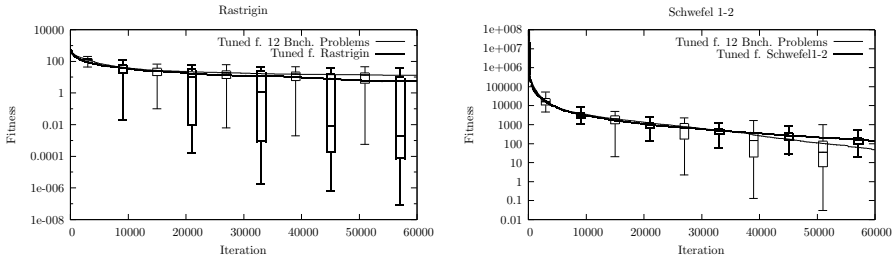

Figure 5.14: Comparison of optimization progress for **PSO-VG** using the behavioural parameters from table 5.6 which were meta-optimized for the benchmark problems individually using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

Variant	Problems	Iterations	S	ω	ϕ_p	ϕ_g
PSO	All Benchmark Problems	60,000	134	-0.1618	1.8903	2.1225
	All Benchmark Problems	600,000	95	-0.6031	-0.6485	2.6475
	Rastrigin & Schwefel1-2	600,000	104	-0.4565	-0.1244	3.0364
PSO-VG	All Benchmark Problems	60,000	198	-0.2723	-	3.8283
	All Benchmark Problems	600,000	134	-0.4300	-	3.0469
	Rastrigin & Schwefel1-2	600,000	72	-0.6076	-	1.9609

Table 5.7: Behavioural parameters for PSO variants that are meta-optimized for various benchmark problems in 30 dimensions and various optimization runlengths. Optimization results are found in figures 5.17-5.20.

Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
Ackley	19.91	0.54	17.18	19.97	20.06	20.15	20.45
Griewank	250.71	100.56	31.99	182.1	228.58	319.37	523.07
Penalized1	14711	35150	15.45	49.56	712.4	7216	157746
Penalized2	381735	637052	48.69	6672	113321	380088	3.09e+6
QuarticNoise	80.82	43.52	9.55	50.2	78.29	114.79	189.5
Rastrigin	191.45	29.84	127.27	170.3	191.75	212.66	272.71
Rosenbrock	1975	1337	526.53	1084	1486	2394	6040
Schwefel1-2	52333	24786	8620	36792	46006	62961	144736
Schwefel2-21	49.46	24.14	20.08	28.63	37.17	75	98.35
Schwefel2-22	77.25	30.36	26.19	55.63	71.64	94.1	144.34
Sphere	14374	10710	194.03	11021	12358	21358	44688
Step	10054	5492	2270	5417	9389	13777	25426

Table 5.8: Optimization end results for **PSO** using the behavioural parameters from Eq.(5.3) which were **hand-tuned**. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 600,000.

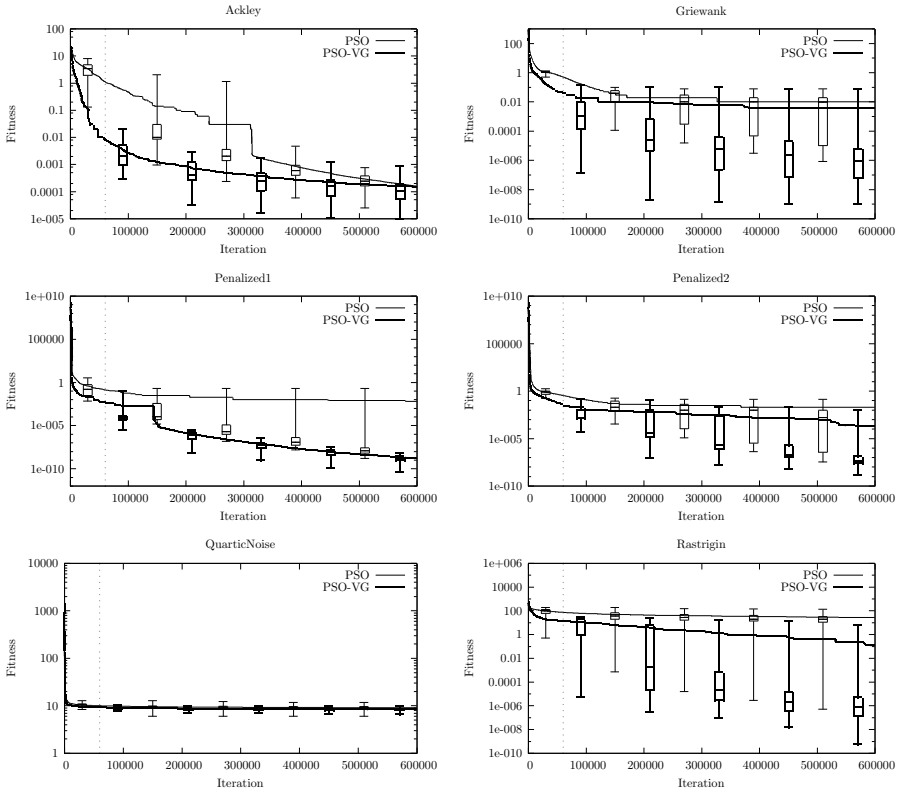

Figure 5.15: Comparison of optimization progress for PSO and PSO-VG using the behavioural parameters from table 5.1, which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization. Dotted line shows 60,000 fitness evaluations for which the parameters were tuned.

Variant	S	ω	ϕ_p	ϕ_g
PSO	167	0.9447	1.4047	0.1494
PSO-VG	195	0.1285	-	0.8231

Table 5.9: Behavioural parameters for **deterministic PSO** variants meta-optimized for all **12 benchmark problems** in 30 dimensions each and optimization run-lengths of **60,000** iterations. Optimization results are found in table 5.11.

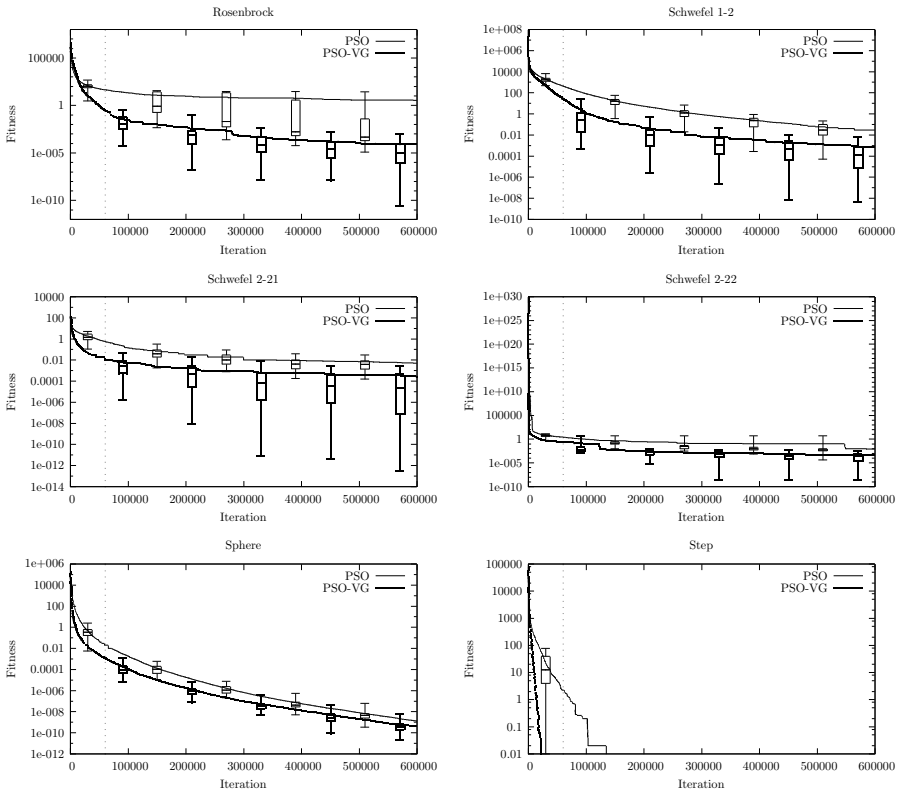

Figure 5.16: Comparison of optimization progress for PSO and PSO-VG using the behavioural parameters from table 5.1, which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization. Dotted line shows 60,000 fitness evaluations for which the parameters were tuned.

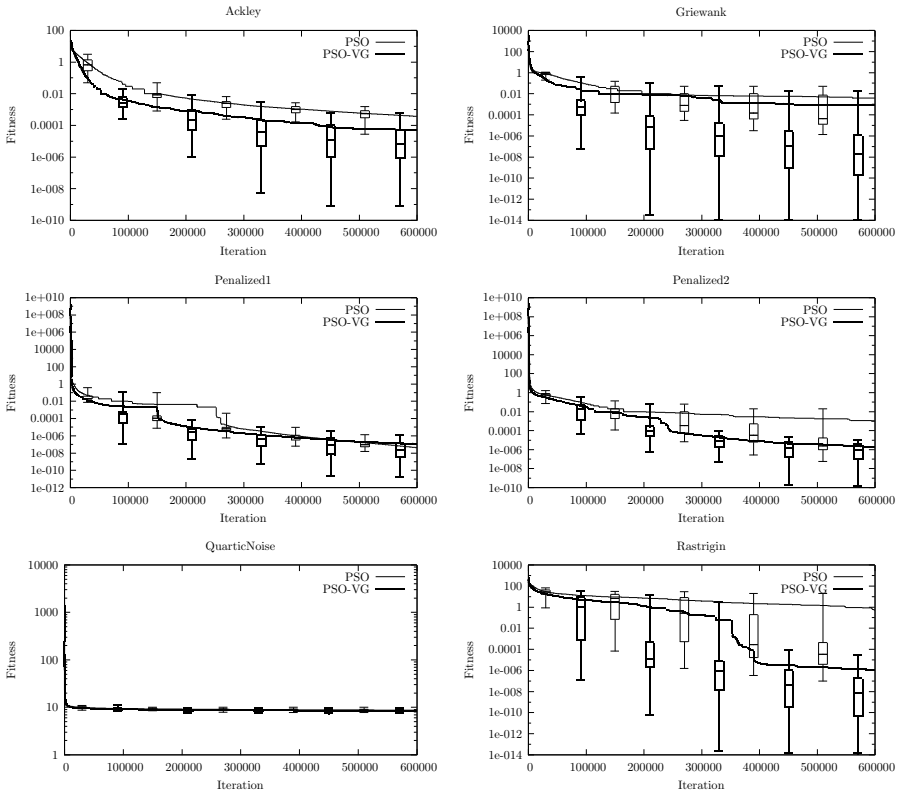

Figure 5.17: Comparison of optimization progress for PSO and PSO-VG using the behavioural parameters from table 5.7, which were meta-optimized for all **12 benchmark problems** using **600,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

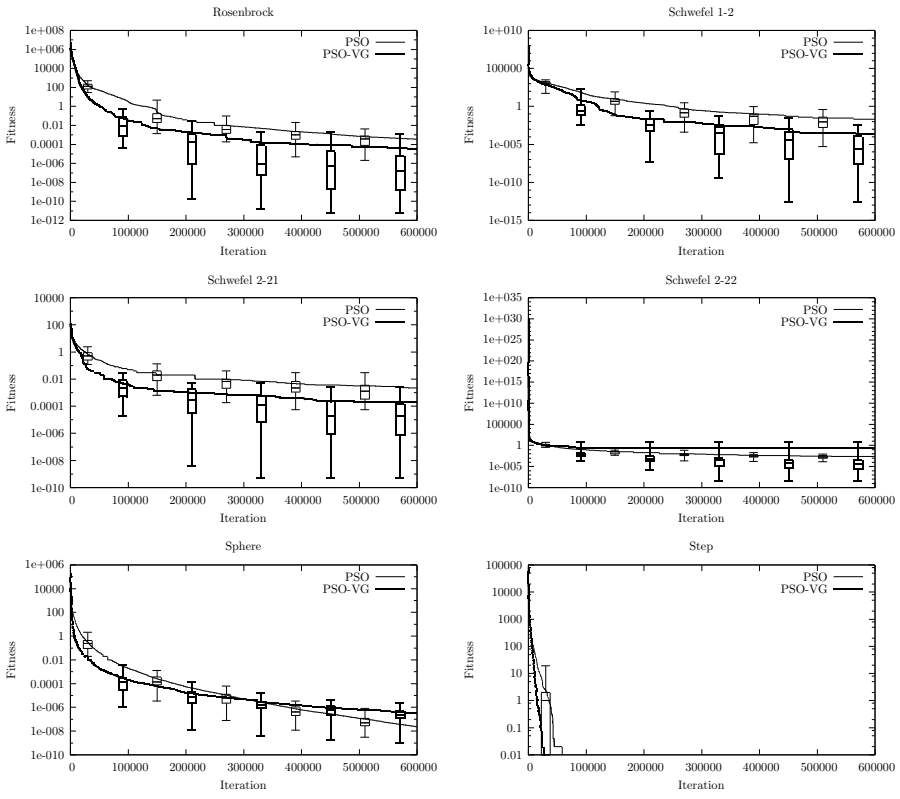

Figure 5.18: Comparison of optimization progress for PSO and PSO-VG using the behavioural parameters from table 5.7, which were meta-optimized for all **12 benchmark problems** using **600,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

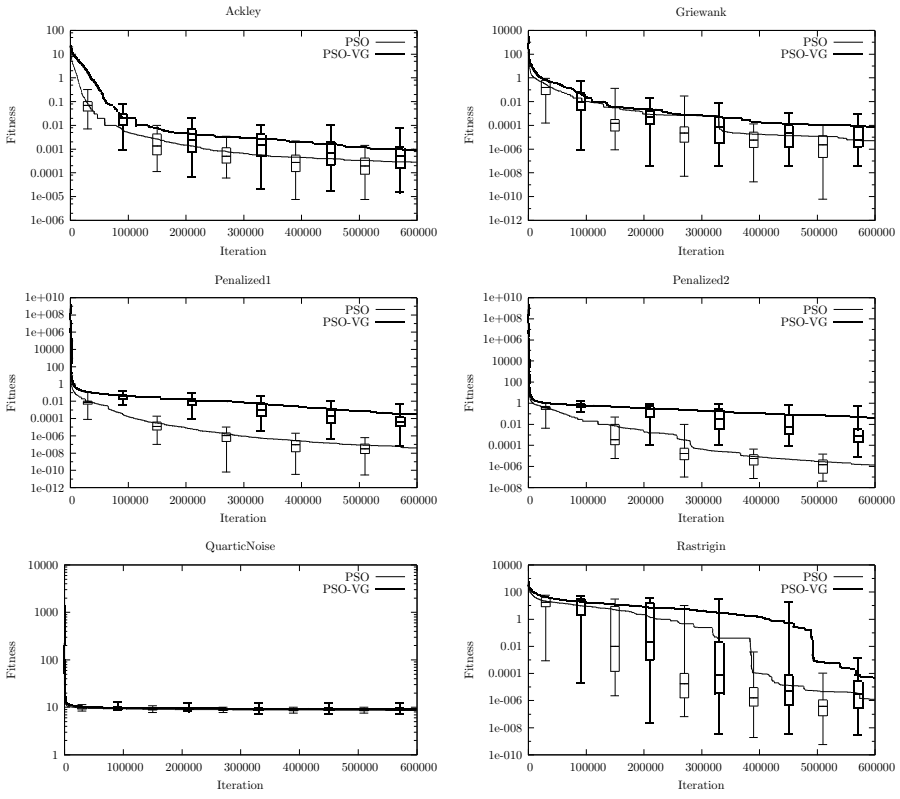

Figure 5.19: Comparison of optimization progress for PSO and PSO-VG using the behavioural parameters from table 5.7, which were meta-optimized for the **Rastrigin & Schwefel1-2** problems using **600,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

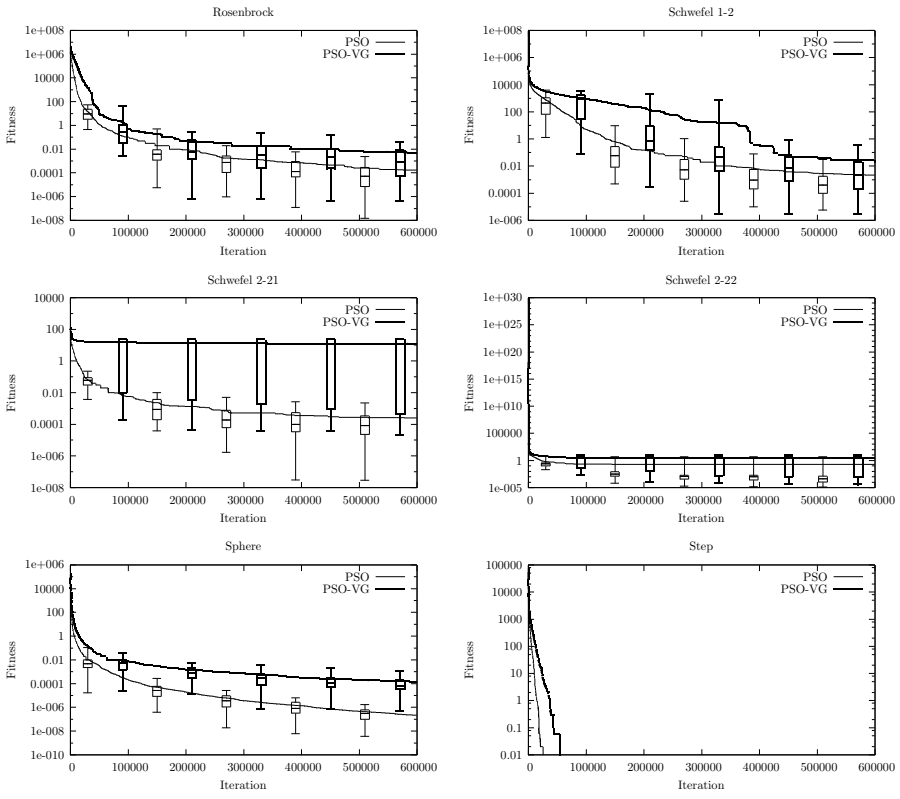

Figure 5.20: Comparison of optimization progress for PSO and PSO-VG using the behavioural parameters from table 5.7, which were meta-optimized for the **Rastrigin & Schwefel1-2** problems using **600,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

		Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
All Benchmark Problems	PSO	Ackley	3.69e-4	2.24e-4	1.87e-5	1.88e-4	3.74e-4	4.89e-4	1.01e-3
		Griewank	3.83e-3	8.76e-3	8.45e-7	5.15e-6	1.74e-5	1.24e-4	0.05
		Penalized1	4.27e-8	4.19e-8	5.97e-9	1.75e-8	3.05e-8	4.99e-8	2.05e-7
		Penalized2	1.1e-3	3.3e-3	3.38e-8	3.05e-7	8.15e-7	2.22e-6	0.01
		QuarticNoise	8.74	0.44	7.8	8.53	8.7	8.97	10.02
		Rastrigin	0.51	2.75	5.54e-8	1.38e-6	9.64e-6	6.97e-5	19.39
		Rosenbrock	3.45e-4	4.34e-4	1.22e-6	4.37e-5	1.85e-4	4.78e-4	2.48e-3
		Schwefel1-2	0.02	0.04	3.02e-6	1.25e-3	4.56e-3	0.03	0.25
		Schwefel2-21	2.26e-3	2.99e-3	5.42e-5	3.17e-4	1.16e-3	2.91e-3	0.02
		Schwefel2-22	2.25e-3	2.06e-3	1.29e-4	5.93e-4	1.64e-3	3.01e-3	7.41e-3
		Sphere	2.37e-8	2.94e-8	8.91e-10	8.22e-9	1.25e-8	3.43e-8	1.77e-7
		Step	0	0	0	0	0	0	0
	PSO-VG	Ackley	5.47e-5	1.08e-4	7.76e-10	8.61e-7	6.3e-6	5.61e-5	6.03e-4
		Griewank	8.15e-4	3.41e-3	0	1.83e-10	1.95e-8	1.12e-6	0.02
		Penalized1	1.11e-7	2.13e-7	1.63e-11	3.12e-9	2.2e-8	1.39e-7	1.05e-6
		Penalized2	1.69e-6	2.58e-6	1.4e-10	4.33e-8	6.07e-7	1.93e-6	1.07e-5
		QuarticNoise	8.5	0.51	7.44	8.18	8.4	8.9	9.81
		Rastrigin	9.81e-7	4.6e-6	1.42e-14	2.02e-11	4.1e-9	1.06e-7	3.26e-5
		Rosenbrock	3.31e-5	1.63e-4	6.04e-12	7.44e-10	9.46e-8	3.49e-6	1.15e-3
		Schwefel1-2	2.72e-4	7.18e-4	2.57e-13	2.85e-8	2.82e-6	1.32e-4	3.59e-3
		Schwefel2-21	2.09e-4	4.45e-4	5.02e-10	7.47e-7	1.98e-5	1.58e-4	2.4e-3
		Schwefel2-22	0.2	0.98	4.36e-9	2.51e-6	4.23e-5	3.99e-4	5
		Sphere	3.34e-7	3.79e-7	9.46e-10	9.9e-8	1.82e-7	4.19e-7	1.94e-6
		Step	0	0	0	0	0	0	0
Rastrigin & Schwefel1-2	PSO	Ackley	2.86e-4	3.22e-4	5.58e-6	5.94e-5	1.27e-4	4.05e-4	1.4e-3
		Griewank	5.17e-6	9.36e-6	4.08e-11	1.98e-7	1.25e-6	4.06e-6	5.32e-5
		Penalized1	3.91e-8	4.88e-8	2.7e-11	4.56e-9	1.38e-8	6.22e-8	1.78e-7
		Penalized2	1.38e-6	1.73e-6	1.34e-8	8.59e-8	7.61e-7	2.16e-6	6.93e-6
		QuarticNoise	8.62	0.51	7.56	8.38	8.51	8.93	10.02
		Rastrigin	1.24e-6	2.69e-6	4.69e-10	4.91e-8	2.69e-7	6.41e-7	1.32e-5
		Rosenbrock	1.71e-4	3.03e-4	1.36e-8	5.56e-6	3.44e-5	1.13e-4	1.39e-3
		Schwefel1-2	2.13e-3	4.98e-3	2.82e-6	3.26e-5	3.04e-4	1.54e-3	0.02
		Schwefel2-21	2.52e-4	4.33e-4	2.96e-8	1.81e-5	7.08e-5	2.33e-4	2.22e-3
		Schwefel2-22	0.2	0.98	1.07e-5	8.67e-5	3.08e-4	1.01e-3	5
		Sphere	2.18e-7	2.11e-7	1.86e-9	5.55e-8	1.46e-7	3.21e-7	8.65e-7
		Step	0	0	0	0	0	0	0
	PSO-VG	Ackley	9.29e-4	1.28e-3	1.49e-5	1.54e-4	5.04e-4	1.18e-3	7.61e-3
		Griewank	7.31e-5	1.58e-4	2.75e-8	1.62e-6	4.96e-6	7.09e-5	9.35e-4
		Penalized1	3.18e-4	7.97e-4	5.94e-8	9.23e-6	3.51e-5	1.51e-4	4.8e-3
		Penalized2	0.04	0.1	7.31e-6	1.58e-4	5.4e-4	2.48e-3	0.4
		QuarticNoise	9.16	0.79	7.28	8.72	9.09	9.47	12.14
		Rastrigin	4.69e-5	1.43e-4	3.11e-9	2.85e-7	2.95e-6	1.44e-5	8.83e-4
		Rosenbrock	5.04e-3	8.28e-3	4.1e-7	5.44e-5	7.27e-4	7.1e-3	0.04
		Schwefel1-2	0.02	0.05	3.11e-6	2.01e-4	2.21e-3	0.02	0.33
		Schwefel2-21	12.01	12.47	2.19e-5	4.25e-4	0.29	25	25.01
		Schwefel2-22	2.9	3.18	3.38e-5	6.97e-4	2.51	5	10
		Sphere	1.46e-4	2.13e-4	4.67e-7	3.01e-5	5.95e-5	2.05e-4	1.02e-3
		Step	0	0	0	0	0	0	0

Table 5.10: Optimization end results for PSO variants using the behavioural parameters from table 5.7 which were meta-optimized for all **12 benchmark problems** and the **Rastrigin & Schwefel1-2** problems, respectively, using **600,000** fitness evaluations. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 600,000.

		Problem	Mean	Std.Dev.	Min	Q1	Median	Q3	Max
PSO	Stochastic	Ackley	1.54	1.33	0.03	0.19	1.35	2.66	4.42
		Griewank	0.54	0.26	5.13e-3	0.36	0.58	0.74	0.98
		Penalized1	0.13	0.28	8.19e-4	2.03e-3	5.57e-3	0.13	1.29
		Penalized2	0.29	0.25	0.01	0.12	0.23	0.41	1.35
		QuarticNoise	10.12	0.75	8.83	9.6	10.06	10.55	12.64
		Rastrigin	81	56.64	0.09	38.23	74.61	118.5	207.86
		Rosenbrock	42.77	29.21	0.14	31.44	35.81	47.67	152.52
		Schwefel1-2	449.19	231.1	1.9	277.51	426.35	593.53	982.47
		Schwefel2-21	0.5	0.47	0.01	0.18	0.27	0.75	1.93
		Schwefel2-22	2.65	3.03	0.16	0.65	1.14	5.19	15.5
		Sphere	0.03	0.04	1.23e-3	6.81e-3	0.02	0.04	0.21
		Step	0.88	1.86	0	0	0	1	7
	Deterministic	Ackley	20.02	1.92	12.6	20.12	20.92	21	21.08
		Griewank	3.08	1.6	1.55	2.02	2.53	3.49	9.79
		Penalized1	15.12	10.01	1.63	8.28	11.44	19.06	43.12
		Penalized2	41.1	16.53	7.32	27.59	44.18	52.71	76.5
		QuarticNoise	16.18	5.11	10.96	12.86	14.16	17.19	33.69
		Rastrigin	281.22	34.37	212.65	261.49	277.5	304.44	356.34
		Rosenbrock	84.07	60.58	36.82	47.7	65.67	115.18	426.44
		Schwefel1-2	2121	1115	453.81	1353	1825	2651	5170
		Schwefel2-21	37.85	14.1	19.33	30.45	35.81	40.24	96.52
		Schwefel2-22	49.88	38.06	12.13	21.74	31.59	78.09	151.46
		Sphere	142.78	123.86	10.72	57.9	106.15	173.51	516.83
		Step	231.5	105.07	69	166	214	291	527
PSO-VG	Stochastic	Ackley	0.01	0.01	4.03e-4	3.8e-3	6.98e-3	0.01	0.05
		Griewank	0.05	0.08	5.09e-6	1.92e-3	8.11e-3	0.07	0.5
		Penalized1	4.74e-3	0.02	1.43e-5	2.15e-4	4.12e-4	6.86e-4	0.11
		Penalized2	0.06	0.08	6.38e-5	0.01	0.03	0.06	0.45
		QuarticNoise	9.27	0.56	8.11	8.89	9.25	9.64	10.93
		Rastrigin	13	11.54	3.5e-4	2.18	12.5	18.04	45.01
		Rosenbrock	0.29	0.67	4.3e-4	8.13e-3	0.04	0.2	3.63
		Schwefel1-2	46.9	97.87	0.02	1.5	10.69	49.83	632.05
		Schwefel2-21	0.01	0.02	2.99e-5	2.73e-3	5.68e-3	0.02	0.15
		Schwefel2-22	0.24	0.97	5.85e-4	9.31e-3	0.03	0.06	5.02
		Sphere	1.07e-3	1.54e-3	4.56e-7	2.4e-4	6.01e-4	1.4e-3	9.93e-3
		Step	0	0	0	0	0	0	0
	Deterministic	Ackley	20.57	0.14	20.25	20.5	20.58	20.66	20.85
		Griewank	451.91	78.61	314.18	393.87	450.31	498.59	657.56
		Penalized1	2.09e+6	2.94e+6	2810	156200	459724	2.72e+6	1.21e+7
		Penalized2	6.52e+6	8.31e+6	54514	2.36e+6	3.97e+6	7e+6	5e+7
		QuarticNoise	109.53	36.41	55.95	84.4	103.11	131.94	230.35
		Rastrigin	258.64	29.31	196.65	241.65	254.35	282.78	329.07
		Rosenbrock	77658	46643	15449	41391	66145	116558	221552
		Schwefel1-2	572000	378879	57334	254583	504665	760026	1.47e+6
		Schwefel2-21	103.27	6.81	83.85	99.94	103.83	108.44	115.42
		Schwefel2-22	219.73	21.62	178.12	204.37	220.2	236.7	273.33
		Sphere	23380	5163	10557	19442	23656	28094	30903
		Step	9312	2255	5401	7739	8976	10813	15945

Table 5.11: Optimization end results for stochastic and deterministic PSO variants using the behavioural parameters from tables 5.1 and 5.9 which were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Table shows end results on benchmark problems of 30 dimensions each, results obtained over 50 optimization runs where the number of fitness evaluations for each run is 60,000.

		PSO					PSO-VG			
		S	ω	ϕ_p	ϕ_g	Meta-Fitness	S	ω	ϕ_g	Meta-Fitness
All Benchmark Problems	60,000 Iterations	134	-0.1618	1.8903	2.1225	2477	198	-0.2723	3.8283	191
		58	-0.1815	-2.6178	2.2893	2520	124	-0.2989	3.6516	192
		63	-0.1734	-2.7807	2.2399	2597	127	-0.3008	3.6689	196
		137	-0.1870	1.7507	2.0190	2620	198	-0.2706	3.9574	211
		67	-0.2683	-2.6941	2.2765	2640	130	-0.3295	3.4083	307
		73	-0.1660	-2.8456	2.4932	2681	198	-0.3126	3.9352	357
		75	-0.1680	-2.4819	2.1990	2803	141	-0.4295	3.1463	539
		57	-0.0682	-2.9241	2.4741	3221	142	-0.5229	2.6741	2805
		35	-0.1596	-3.2256	3.4187	3891	122	-0.1655	3.4526	3013
		29	-0.1015	-3.4645	2.9721	3925	173	-0.5148	3.0156	5119
	600,000 Iterations	95	-0.6031	-0.6485	2.6475	36	134	-0.4300	3.0469	35
		93	-0.6092	-0.5166	2.5289	36	136	-0.4505	2.8850	36
		104	-0.5712	-0.5501	2.9829	37	138	-0.4546	2.8991	36
		98	-0.3892	3.7110	-0.4651	37	135	-0.4027	3.0895	36
		95	-0.3825	3.8995	-0.3487	38	132	-0.4402	2.8853	37
		102	-0.4151	3.6217	-0.3605	38	136	-0.4477	3.0220	37
		100	-0.4758	3.6132	-0.4235	39	125	-0.2962	3.5748	43
		103	-0.4763	3.8797	-0.8518	42	75	-0.6388	1.7474	151
		125	-0.5873	-1.5224	3.7938	42	72	-0.6290	1.6422	157
		167	-0.4713	3.7834	-0.9089	43	180	-0.7571	-2.0636	5526
Rastrigin & Schwefel1-2	60,000 Iterations	82	-0.3794	-0.2389	3.5481	971	47	-0.3000	3.5582	1013
		79	-0.3617	-0.3757	3.6127	1247	91	-0.3128	3.6807	1027
		78	-0.3475	-0.2667	3.5363	1567	89	-0.3096	3.5499	1247
		78	-0.3960	-0.3805	3.5699	2628	81	-0.2681	3.8548	1328
		128	-0.1739	-2.8805	2.6971	10850	107	-0.3971	2.9703	1490
		129	-0.2018	-2.7798	2.8136	10950	137	-0.3797	3.1818	1618
		129	-0.1793	-2.9783	2.779	11770	112	-0.3816	3.1116	1646
		131	-0.2298	-2.7022	2.8644	12364	144	-0.4144	2.9859	1683
		130	-0.1761	-2.7207	2.9896	12624	105	-0.2731	3.4445	1830
		137	-0.0750	-2.4195	3.0308	16080	93	-0.2285	3.7267	2198
	600,000 Iterations	104	-0.4565	-0.1244	3.0364	0.04	72	-0.6076	1.9609	0.86
		102	-0.4509	-0.1825	3.0732	0.07	67	-0.5900	1.9807	9.27
		106	-0.5275	-0.3622	3.0008	0.07	108	-0.0153	3.5932	34078
		109	-0.5444	-0.2456	2.7428	0.13	132	-0.7774	-2.0419	51456
		103	-0.5447	-0.571	3.5057	0.36	90	-1.0630	-1.8378	56003
		95	-0.3143	2.1649	1.4668	1.72	131	-0.7488	-2.0159	66937
		92	-0.2907	2.2996	1.5602	5.14	126	-0.7474	-2.2825	76404
		92	-0.2701	2.2933	1.5871	19.49	125	-0.8157	-1.9518	78618
		182	-0.3354	-2.7778	2.053	289.99	121	-0.7931	-1.9929	80899
		191	-0.3765	-2.8861	2.3334	314.56	118	-0.9791	-1.8073	83891

Table 5.12: Best 10 sets of behavioural parameters for PSO variants that are meta-optimized for various combinations of benchmark problems and optimization run-lengths.

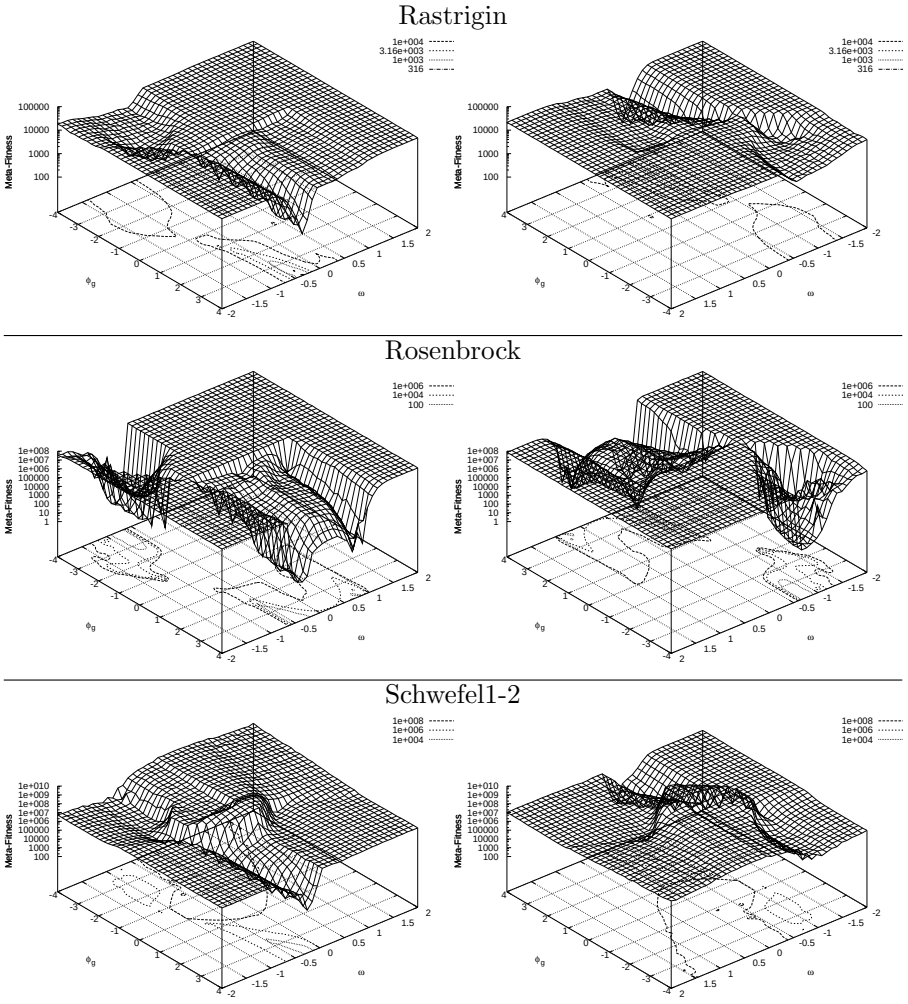

Figure 5.21: Meta-fitness landscape shown at different angles for **PSO-VG** computed by varying the ω and ϕ_g parameters and keeping a fixed swarm-size $S = 100$, measuring the performance of PSO-VG on **Rastrigin**, **Rosenbrock** and **Schwefel1-2** in 30 dimensions over 50 optimization runs of **60,000** fitness evaluations each.

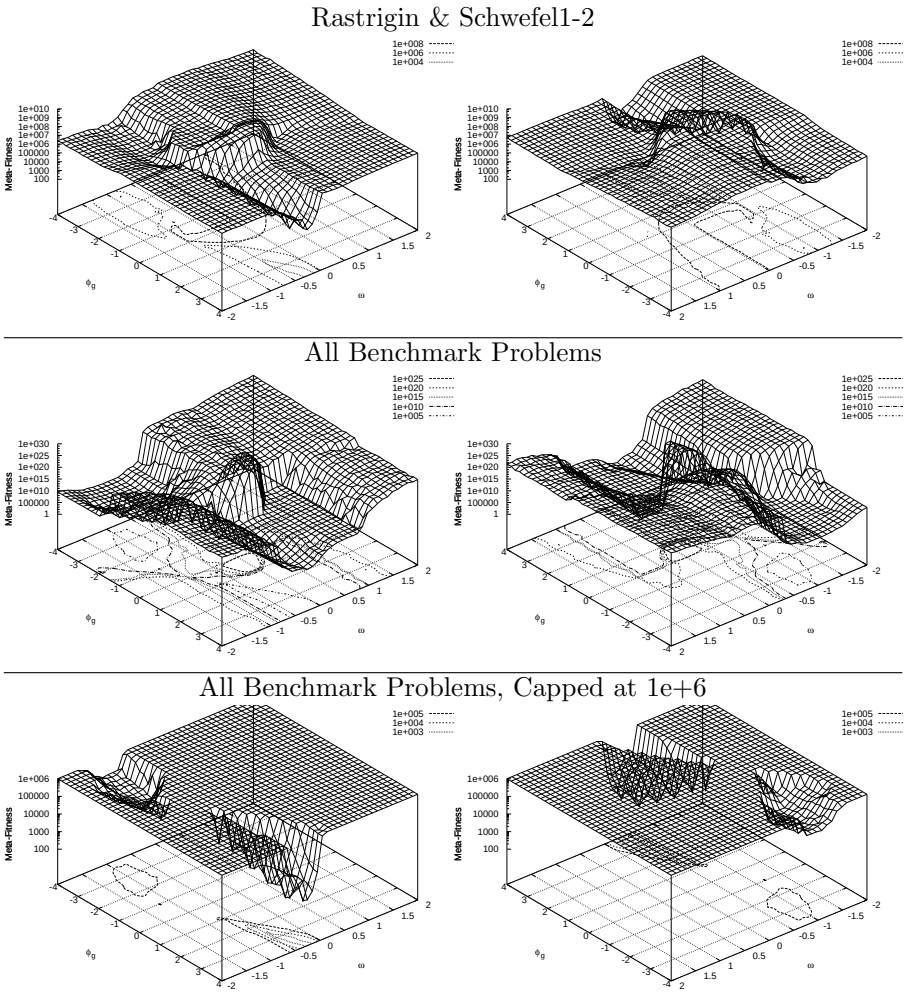

Figure 5.22: Meta-fitness landscape shown at different angles for **PSO-VG** computed by varying the ω and ϕ_g parameters and keeping a fixed swarm-size $S = 100$, measuring the performance of PSO-VG on respectively **Rastrigin & Schwefel1-2** and all **12 benchmark problems** in 30 dimensions over 50 optimization runs of **60,000** fitness evaluations each.

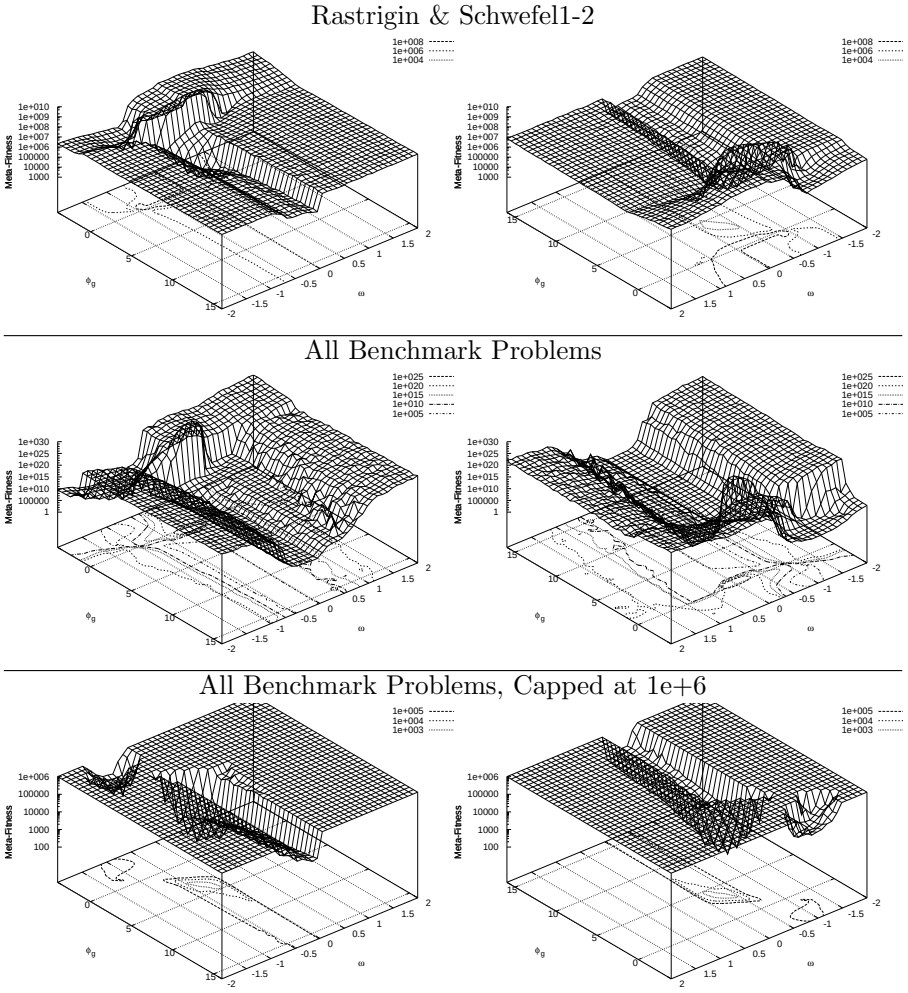

Figure 5.23: Meta-fitness landscape shown at different angles for **PSO-VG** computed by varying the ω and ϕ_g parameters and keeping a fixed swarm-size $S = 200$, measuring the performance of PSO-VG on respectively **Rastrigin & Schwefel1-2** and all **12 benchmark problems** in 30 dimensions over 50 optimization runs of **60,000** fitness evaluations each. Compared to figure 5.22 these plots are for a swarm-size of $S = 200$ instead of $S = 100$ and a **wider boundary** for ϕ_g.

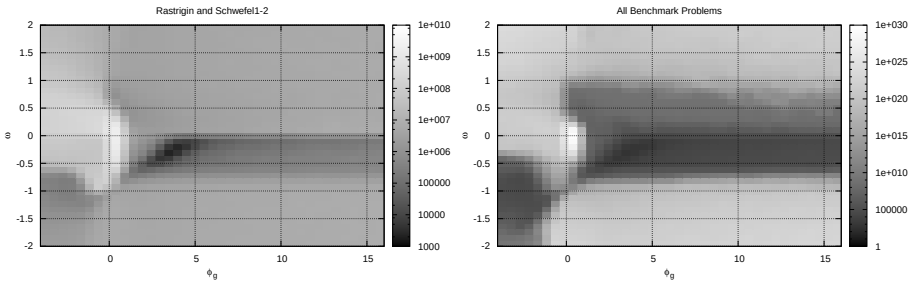

Figure 5.24: Depth of meta-fitness landscape for **PSO-VG** computed by varying the ω and ϕ_g parameters and keeping a fixed swarm-size $S = 200$, measuring the performance of PSO-VG on respectively **Rastrigin & Schwefel1-2** and all **12 benchmark problems** in 30 dimensions over 50 optimization runs of **60,000** fitness evaluations each. Darker grey means better performance, so the region around $\omega \simeq -0.3$ and $\phi_g \simeq 3.5$ seems to be best in both plots.

S	ω	ϕ_g	Meta-Fitness
130	-0.4135	3.1937	4.01e-3
216	-0.2531	4.1917	4.01e-3
191	-0.4685	2.8533	0.01
185	-0.4597	2.9226	0.02
212	-0.3027	4.2061	0.02
247	-0.4522	2.9134	0.02
198	-0.4313	2.9630	0.03
194	-0.3620	3.2545	0.03
218	-0.4888	2.6767	0.05
248	-0.4615	2.7601	0.09

Table 5.13: Best 10 sets of behavioural parameters for **PSO-VG** that are meta-optimized for **Rastrigin & Schwefel1-2** in 30 dimensions each and optimization run-lengths of **600,000** iterations. Compared to table 5.12 these have been meta-optimized with wider boundaries for the S and ϕ_g parameters.

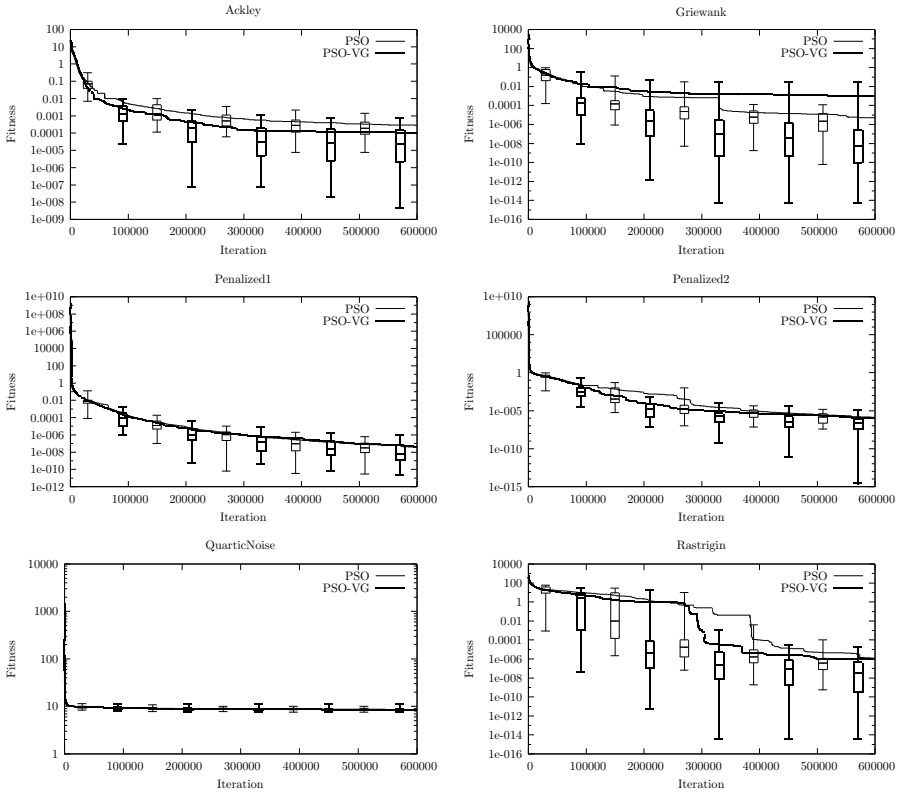

Figure 5.25: Comparison of optimization progress for PSO and PSO-VG using the behavioural parameters for PSO from table 5.7 and the best parameters for PSO-VG from table 5.13, which were meta-optimized for the **Rastrigin & Schwefel1-2** problems using **600,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

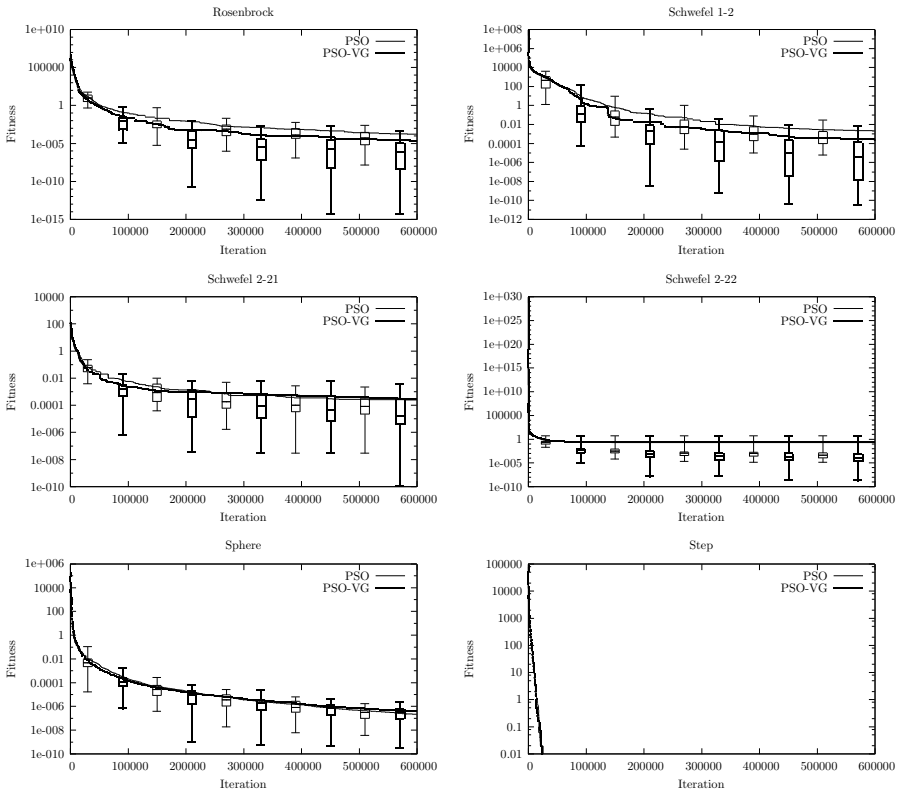

Figure 5.26: Comparison of optimization progress for PSO and PSO-VG using the behavioural parameters for PSO from table 5.7 and the best parameters for PSO-VG from table 5.13, which were meta-optimized for the **Rastrigin & Schwefel1-2** problems using **600,000** fitness evaluations. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

Problems	Optimization Iterations	Time Usage	
		PSO	PSO-VG
Rosenbrock & Sphere	60,000	27 min	25 min
Rastrigin & Schwefel1-2	600,000	10 h 26 min	6 h 19 min
QuarticNoise, Sphere & Step	60,000	1 h 45 min	1 h 16 min
All 12 Benchmark Problems	60,000	3 h 38 min	2 h 46 min
All 12 Benchmark Problems	600,000	43 h 11 min	27 h 26 min

Table 5.14: Time usage for meta-optimizing the behavioural parameters of **PSO** and **PSO-VG** with various combinations of benchmark problems and optimization iterations being used.

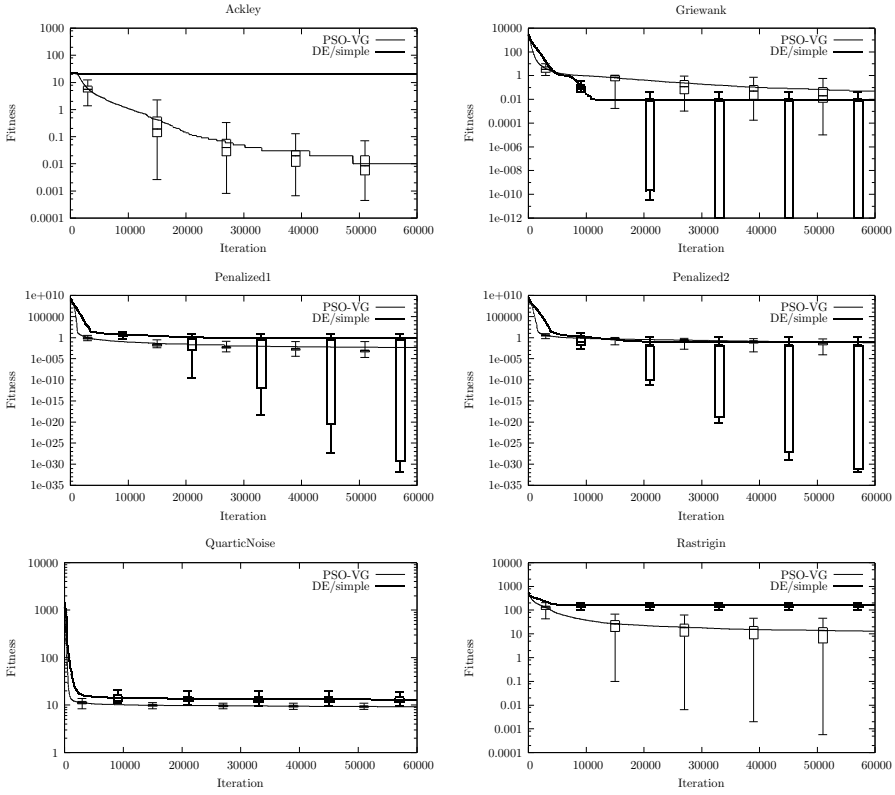

Figure 5.27: Comparison of optimization progress for **PSO-VG** and **DE/Simple** using behavioural parameters that were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Reprinted from figures 5.3-5.4 and 4.7-4.8. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

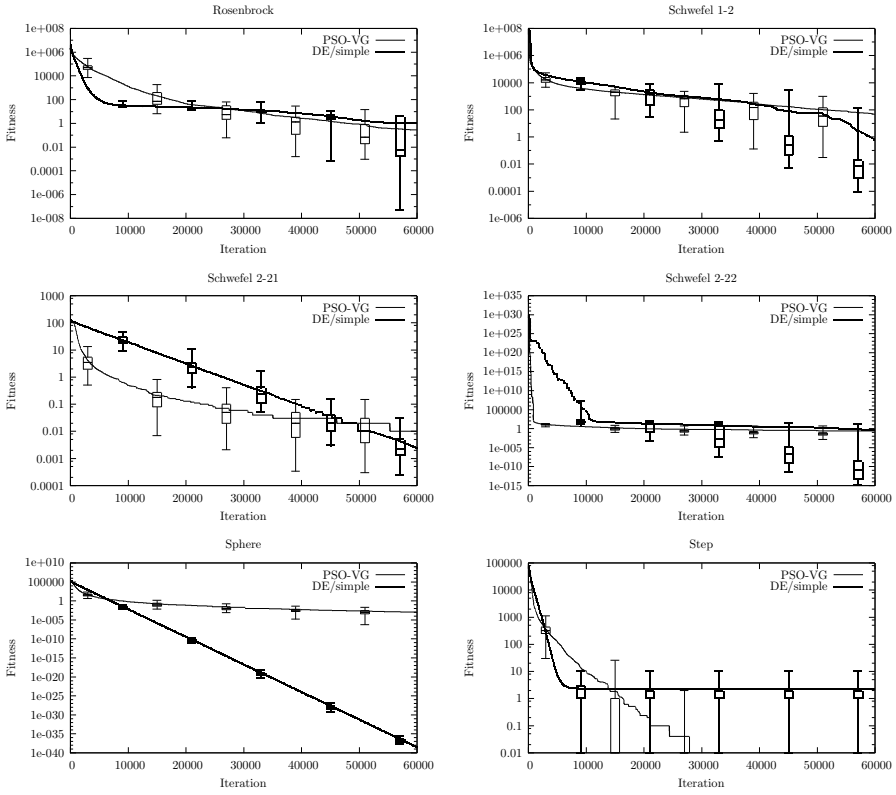

Figure 5.28: Comparison of optimization progress for **PSO-VG** and **DE/Simple** using behavioural parameters that were meta-optimized for all **12 benchmark problems** using **60,000** fitness evaluations. Reprinted from figures 5.3-5.4 and 4.7-4.8. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

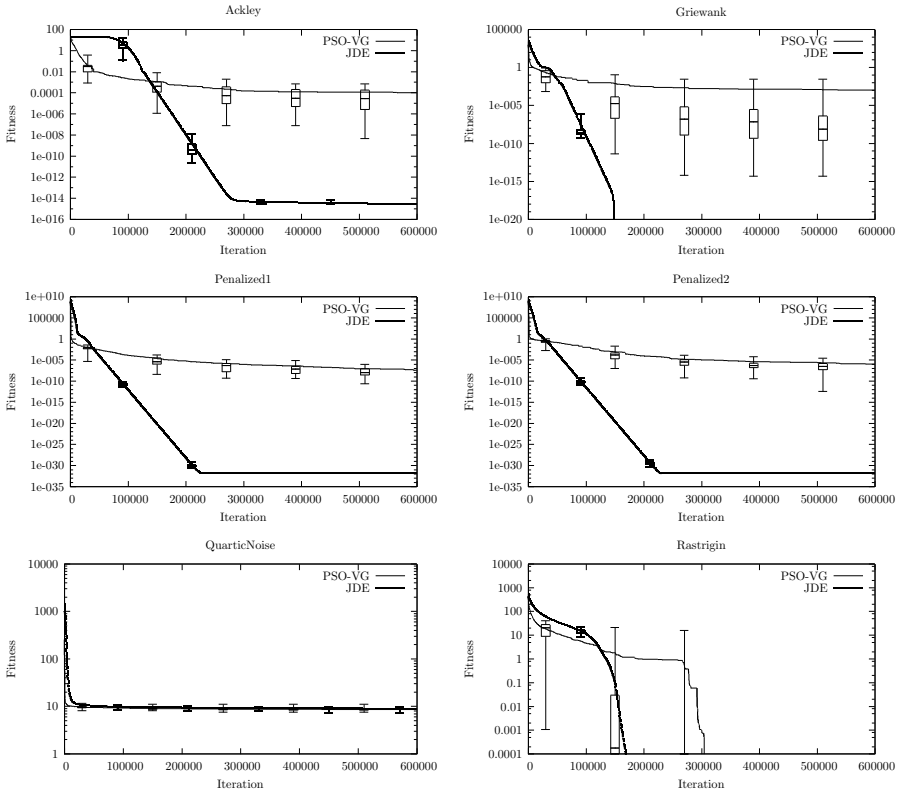

Figure 5.29: Comparison of optimization progress for **PSO-VG** and **JDE/rand/1/bin** using behavioural parameters that were meta-optimized for **Rastrigin & Schwefel1-2** (PSO-VG) and **Ackley, Rastrigin, Rosenbrock & Schwefel1-2** (JDE) when using **600,000** fitness evaluations. Reprinted from figures 5.25-5.26 and 4.29-4.29. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

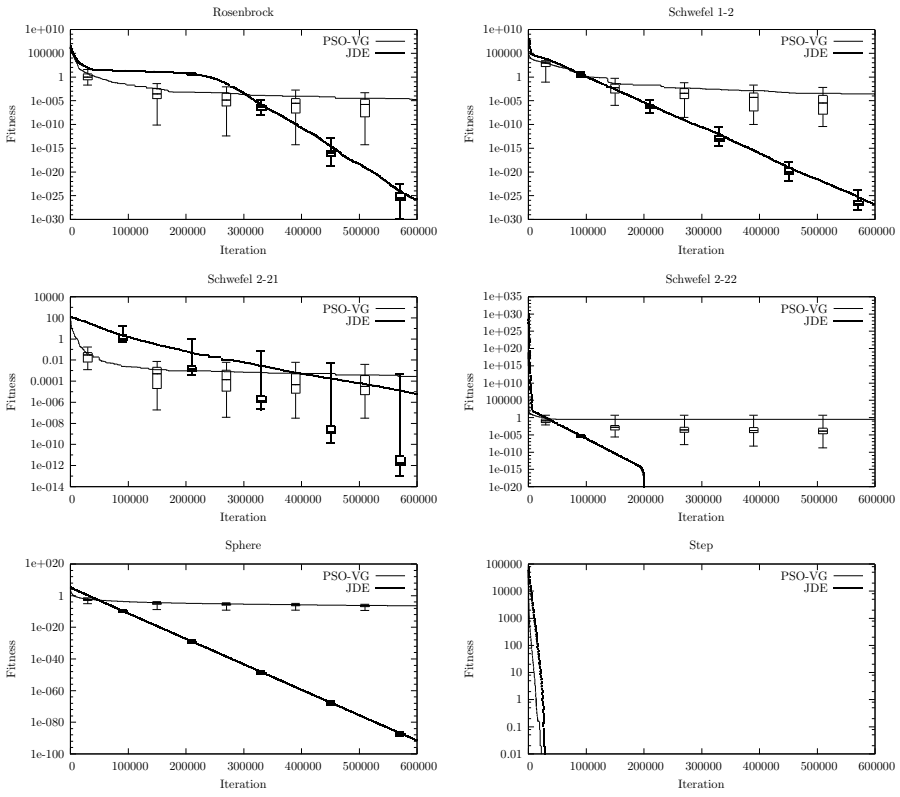

Figure 5.30: Comparison of optimization progress for **PSO-VG** and **JDE/rand/1/bin** using behavioural parameters that were meta-optimized for **Rastrigin & Schwefel1-2** (PSO-VG) and **Ackley, Rastrigin, Rosenbrock & Schwefel1-2** (JDE) when using **600,000** fitness evaluations. Reprinted from figures 5.25-5.26 and 4.29-4.29. Plots show the mean fitness achieved over 50 optimization runs, as well as the quartiles at intervals during optimization.

Chapter 6

Conclusion

6.1 Main Contributions

6.1.1 Meta-Optimization

Optimization methods often have behavioural parameters the user must select to achieve good performance. Even if practitioners will only be using the default parameters provided by the author of an optimization method, the author must still do significant testing of different parameter choices to find the ones that perform well and also generalize to new optimization problems. Choosing the parameters can be done by manual experimentation by a human researcher, or by exhaustive search of combinations of parameters, or by using an overlaid Meta-Optimizer to search for good parameters. Manual experimentation uses an expensive and scarce resource, human thought, while grid-based search of parameter combinations is impossible for more than a few behavioural parameters due to the Curse of Dimensionality and the exponential increase in computation time with more parameters. On the other hand, meta-optimization has been shown to be comparatively cheap to execute yet superior in the optimization performance of the tuned parameters. Because of this, meta-optimization can also lead to new insight into what makes an optimizer work, as many experiments with tuning the behavioural parameters for different scenarios can be made with little effort. Moreover, the technique presented in chapter 3 for doing meta-optimization is simple to describe and implement and is therefore useful for both experienced researchers and beginners.

6.1.2 Adaptive Vs. Simplified Optimizers

Another and much more popular approach in the research literature for selecting the behavioural parameters of an optimizer is to try and adapt the parameters during optimization. It was demonstrated in the experiments of chapters 4 and 5 that there is no general and consistent advantage in doing so-called parameter adaptation. There are advantages in certain optimization scenarios but they

are countered by disadvantages elsewhere. It was also demonstrated that optimizers may be simplified with improved performance in several cases, and their behavioural parameters became easier to tune as well.

6.2 Recommendations for Future Research

6.2.1 Boundaries For Behavioural Parameters

This thesis used boundaries for defining the search-space of behavioural parameters, but they were really just meant as a guidance for which combinations of behavioural parameters might be interesting. Sometimes the boundaries turned out to be too narrow and the meta-optimization experiments had to be redone with wider boundaries for the behavioural parameters, which was time consuming. The obvious suggestion is to use an unbounded version of LUS as the meta-optimizer, which, however, will need some research as LUS may not perform as well in an unbounded search-space as it does in a bounded one.

6.2.2 Meta-Fitness Landscape Approximation

Even though the simple technique of Preemptive Fitness Evaluation was able to greatly reduce the time-usage involved in doing meta-optimization it still remains a computationally expensive task. To save even more time in meta-optimization one could approximate the meta-fitness landscape, so as to try and predict meta-fitness values for previously unseen parameter combinations. This was done in various ways by Ridge and Kudenko [136], Bartz-Beielstein et al. [137] and Smit and Eiben [33].

The challenge with using an approximator in meta-optimization is that very few sample mappings are available. Another challenge is to combine both meta-fitness approximation and the technique of Preemptive Fitness Evaluation, as the latter causes many of the meta-fitness measures to be incomplete due to their evaluations being preemptively aborted, which might disrupt the approximator in accurately mimicking the meta-fitness landscape.

6.2.3 Parallelization & Distributed Computation

Another approach to save time in meta-optimization would be to distribute the execution to several computational nodes. Since the LUS method used here as meta-optimizer only has a single optimizing agent it will take some research to find a good approach for distributing its computation. One suggestion would be to make a multi-agent version of LUS where each agent is distributed to a computation node, and then synchronize update of the best behavioural parameters providing the center from which all new parameters are sampled.

6.2.4 Other Meta-Fitness Measures

The experiments here with meta-optimization all used a meta-fitness measure based on the average fitness achieved by an optimizer using a certain number of iterations. Other useful performance measures to tune for could be the number of optimization iterations required to achieve a certain goal, or perhaps the rate of optimization as measured by the integral of the fitness progress. Some meta-fitness measures may not be supported by Preemptive Fitness Evaluation, however, and will therefore take significantly more time to meta-optimize.

6.2.5 Multi-Objective Meta-Optimization

This thesis has focused on single-objective optimization problems and meta-optimization was therefore also a single-objective task. Many optimization problems occurring in the real world are multi-objective by nature and tuning the behavioural parameters of a multi-objective optimization method is therefore Multi-Objective Meta-Optimization. It is currently unclear what would be the best approach for doing this, without sacrificing the simplicity or efficacy of the approach that was presented here for doing single-objective meta-optimization. One suggestion would be to make LUS multi-objective by replacing its comparison operator with the Pareto domination operator from Eq.(3.1). Additionally, the tuned parameters should cause the optimizer to have good coverage of the Pareto front and perhaps this could be achieved by taking a measure of spread into account when LUS selects which behavioural parameters are best.

6.2.6 Meta-Meta-Optimization

The LUS method was used as meta-optimizer with its own behavioural parameter being found by manual experimentation. It would be interesting to see if this behavioural parameter of LUS itself could be tuned to make LUS perform even better when used as a meta-optimizer. If LUS is going to be used often as the meta-optimizer then it makes perfect sense to find the behavioural parameter that makes it perform its best at this task. This would effectively mean that the behavioural parameter of the LUS method should be *Meta-Meta-Optimized*, and this naturally raises the question of which optimization method to use as the meta-meta-optimizer? Since the meta-meta search-space can still be expected to be fairly smooth, a suggestion would be to use the LUS method again, and with its standard parameter that worked well for ordinary meta-optimization.

Although this may seem silly to people who are just beginning to accept the usefulness of doing meta-optimization, comparisons of techniques for doing meta-optimization have actually already been made in the literature to identify which approach works best, which is a manual way of doing meta-meta-optimization, see Smit and Eiben [33]. Indeed, this was also done in chapter 3 here.

Meta-Meta-Meta-...

Does it then make sense to tune the behavioural parameter of LUS when it is being used as the meta-meta-optimizer? This does not appear to be the case, because the parameter search-spaces for each additional meta-layer seem to become increasingly simple and smooth and when we reach the meta-meta-layer they just might be smooth enough to almost guarantee that the best performing parameters can be found.

6.2.7 Evolving An Optimization Method

Meta-optimization was used in this thesis to tune the behavioural parameters of an optimization method. An interesting idea would be to not only tune the parameters but also the actual optimization algorithm. This has already been studied to some extent for evolving specialized optimizers by Bengio et al. [138] and Radi and Poli [139], and for evolving PSO variants by Poli et al [140]. They used Genetic Programming (GP) which employs basic evolutionary concepts to construct computer programs (see chapter 1).

A more general way of evolving actual optimization algorithms would be to first define a language for concisely describing optimization algorithms and then use an overlaying optimization method akin to GP but tailored to work on strings from this language. The concept of evolving strings from a given language has been studied by O'Neill et al. [141] [142] [143] who called it *Grammatical Evolution* (GE) and works on arbitrarily long strings of integers, which are then translated to syntactic trees according to some predefined grammar and whose fitness can then be computed. An extension to GE is due to Ortega et al. [144] and ensures the evolved strings are not only syntactically correct but also semantically correct.

The language that is used for describing optimization algorithms can itself be evolved so as to be more expressive. This can be achieved by applying GE in an overlaying meta-manner and preliminary studies of this are reported by O'Neill and Ryan [145].

6.2.8 Bootstrapped Evolution of Optimization Methods

Taking the idea of evolving an optimization algorithm one step further it can be used in a bootstrapped manner to gradually improve itself. The idea is once again to start by defining (or evolving) a language for concisely describing optimization algorithms. Then implement a simple optimization method in this language that works on instances of the same language. The method can then be used for optimizing itself, perhaps ad infinitum. A similar idea for general problem solving is proposed in the *Gödel* machine by Schmidhuber [146] [147].

Appendix A

Non-Convergence Analysis

A.1 Introduction

This appendix contains simple mathematical analysis that proves an optimizer which does not somehow adapt its search- or sampling-range cannot converge to a local optimum in a timely manner.

A.2 The Sphere Function

The Sphere benchmark function from chapter 2 will be used in this analysis due to its simplicity, thus leading to graceful mathematical derivations. Furthermore, because the Sphere function is perhaps the simplest optimization problem available that is both continuous and unimodal, it seems reasonable that when an optimization method fails at optimizing the Sphere function, then the method will probably also fail in optimizing more difficult problems.

The Sphere function has also been used by other researchers for analyzing convergence aspects of optimization methods, see for example [20] [60] [70].

Sphere Re-Write

First recall the definition of the Sphere function from chapter 2:

$$f(\vec{x}) = \sum_{i=1}^{n} x_i^2$$

and notice that it is actually the dot-product of vector $\vec{x}$ with itself, that is:

$$f(\vec{x}) = \sum_{i=1}^{n} x_i^2 = \vec{x} \cdot \vec{x} = \|\vec{x}\|^2 \tag{A.1}$$

where the last identity is basic linear algebra. The vectors $\vec{x}$ evaluating to a fitness less than some fitness-value $F \geq 0$ must therefore satisfy:

$$f(\vec{x}) \leq F \;\Leftrightarrow\; \|\vec{x}\|^2 \leq F \;\Leftrightarrow\; \|\vec{x}\| \leq \sqrt{F}$$

which designates the closed hyperball of radius $\sqrt{F}$. Let us denote a general n-dimensional hyperball with radius r by $B_n(r)$, thus defined as:

$$B_n(r) = \{\vec{x} \,:\, \|\vec{x}\| \leq r\}$$

with the vectors $\vec{x}$ being n-dimensional. This means positions having a fitness better than or equal to F, must satisfy:

$$f(\vec{x}) \leq F \;\Leftrightarrow\; \vec{x} \in B_n\left(\sqrt{F}\right) \tag{A.2}$$

Expressing this fitness requirement as set-membership will prove convenient in the probability-theoretical analysis of the Sphere function below.

Hyperball Volume

The volume of a hyperball $|B_n(r)|$ is defined in [148, p. 1442] in terms of the constant s_n, which is the hyper-surface area of the n-dimensional hypersphere with unit radius. The hyperball volume is then defined as:[1]

$$|B_n(r)| = \frac{s_n \cdot r^n}{n} \tag{A.3}$$

But what is s_n? Although [148, Eq.(7), p. 1442] provides an explicit formula for computing s_n, it is fairly complicated. Fortunately enough, the value s_n is not needed for the analysis to follow, as the probability boundaries may be studied instead, thus factoring out s_n. And this boundary study actually suffices for the conclusions that are sought.

A.3 Local Sampling

Mathematical analysis is now used to show that optimization methods using local sampling with a fixed sampling-range are actually incapable of converging to a local optimum in a timely manner.

A.3.1 Basic Local Sampling

The essential idea of local sampling is to choose a random point $\vec{y}$ from the neighbourhood of the current position $\vec{x}$ in the search-space, and move to the new position in case of fitness improvement. That is, let the new potential position $\vec{y}$ be defined as:

$$\vec{y} = \vec{x} + \vec{a}$$

[1] Notation adjusted to context.

with $\vec{a}$ chosen randomly and uniformly as follows:

$$\vec{a} \sim U(-\vec{d}, \vec{d})$$

where $\vec{d}$ is the search-range.

But keeping a fixed search-range d throughout the optimization run will now be shown mathematically to decrease the probability of fitness improvement, the closer we get to the optimum of the Sphere function.

A.3.2 Single-Dimensional Case

To understand the underlying idea of this proof, first consider the single dimensional Sphere function:

$$f(x) = x^2, \ x \in \mathbb{R}$$

where any uniformly localized sampling method will calculate the agent's potential new position as:

$$y = x + a$$

with $a \sim U(-d, d)$ for some sampling range delimiter $d > 0$. But now consider x approaching the global optimum, then less and less of the $(-d, d)$ range will result in a fitness improvement of the new randomly chosen y. To see this, recall Eq.(A.1) from which it follows that:

$$f(\vec{y}) < f(\vec{x}) \ \Leftrightarrow \ \|\vec{y}\| < \|\vec{x}\| \tag{A.4}$$

and specifically for the one-dimensional case: $f(y) < f(x) \ \Leftrightarrow \ |y| < |x|$. So a sample y taken from the range $(-|x|, |x|)$ will result in an improvement over x, but since y is taken from the range $(|x| - d, |x| + d)$, it means the probability of finding such an improved position for y, is given by the amount of overlap between these two ranges. For the one-dimensional case, this can be expressed as:

$$\Pr\left[f(y) < f(x)\right] = \begin{cases} |x|/d & \text{, if } |x| < d/2 \\ 1/2 & \text{, else} \end{cases}$$

Initially, when x is far from the global optimum, the probability of finding an improved position by localized random sampling, is always $1/2$. This is demonstrated in figure A.1. But as x approaches the global optimum, and when the sampling range d remains fixed, the probability of improvement approaches zero. This is demonstrated in figure A.2.

As the probability of improvement is non-zero unless x is already situated in the optimum, it is also possible to prove that an improved position y is inevitably discovered if enough samples are taken. So the method does eventually converge to the optimum. But convergence for an optimization method should never be proven using extreme limit-cases, without augmenting that proof with a good estimate on the computational effort required to actually obtain satisfactory results. In the case of local sampling with a fixed range, it was just shown that the computational effort approaches infinity as the optimum is approached.

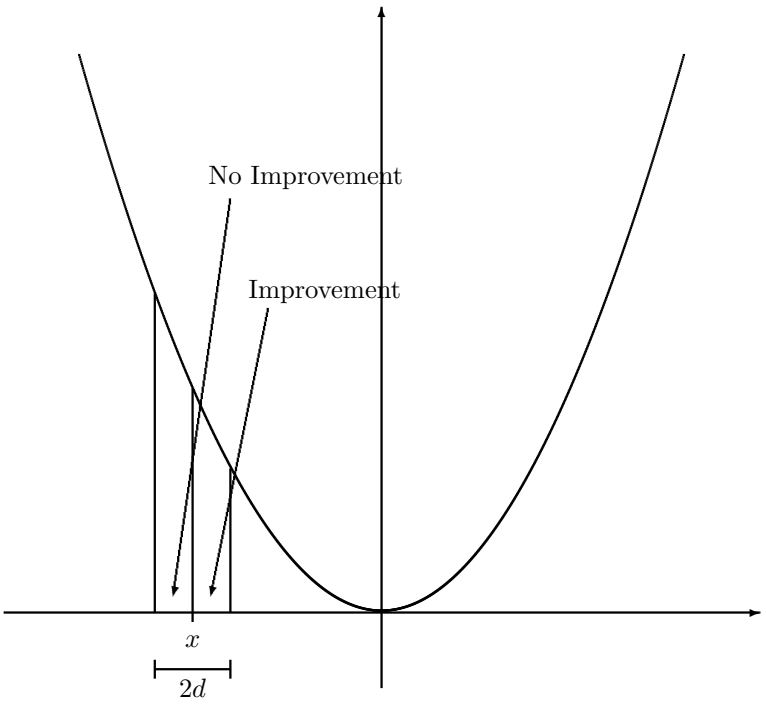

Figure A.1: Local sampling of the Sphere function. When the current position x is far from the optimum, the probability of improvement is always $1/2$.

Since the Sphere function is *separable*, in the sense that the dimensions are independent of each other, this argument can be repeated for each dimension in the multi-dimensional case to show the same result applies there. But it is also possible to show it using set-volumes, as will be done next.

A.3.3 Multi-Dimensional Case

The idea from the one-dimensional Sphere function can be extended directly to n-dimensional search-spaces. The relation from Eq.(A.4) is used again and states the conditions under which some position $\vec{y}$ will improve on the fitness of $\vec{x}$, namely when the length of $\vec{y}$ is shorter than the length of $\vec{x}$.

In multi-dimensional localized sampling, $\vec{y}$ is taken from the hypercube surrounding $\vec{x}$ and having sidelengths $2d$ (assuming the search-range is identical for all dimensions). Let this n-dimensional hypercube be denoted by:

$$C_n(\vec{x}, d) = [x_1 - d,\, x_1 + d] \times \cdots \times [x_n - d,\, x_n + d]$$

So the probability of $\vec{y}$ improving the fitness of $\vec{x}$, is the size of the intersection between this hypercube and the (open) hyperball of radius $\|\vec{x}\|$, divided by the

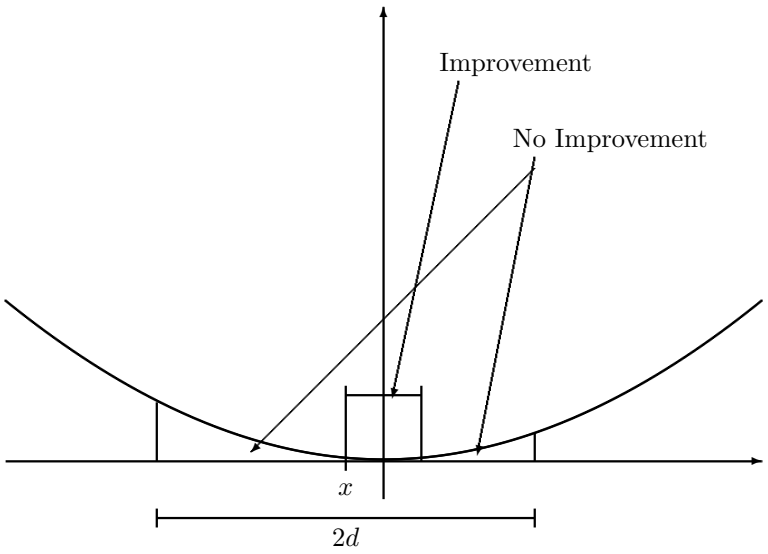

Figure A.2: Local sampling of the Sphere function. When the current position x is close to the optimum, the probability of improvement is $2|x|/(2d) = |x|/d$. If d remains fixed, this probability will approach zero as x approaches the optimum.

total size of the hypercube used for sampling, that is:

$$\Pr\left[f(\vec{y}) < f(\vec{x})\right] = \frac{|C_n(\vec{x}, d) \cap B_n(\|\vec{x}\|)|}{|C_n(\vec{x}, d)|} = \frac{|C_n(\vec{x}, d) \cap B_n(\|\vec{x}\|)|}{(2d)^n} \qquad \text{(A.5)}$$

where the last identity follows from the fact that $|C_n(\vec{x}, d)| = (2d)^n$.

The exact value of Eq.(A.5) is not important, as we only need to note a few things. First, that the denominator $(2d)^n$ remains constant because the range delimiter d of the sampling hypercube is fixed. Second, that the numerator approaches zero as $\vec{x}$ approaches the global optimum of the Sphere function, because $|B_n(\|\vec{x}\|)|$ and hence $|C_n(\vec{x}, d) \cap B_n(\|\vec{x}\|)|$ approach zero. So the probability in Eq.(A.5) approaches zero as we approach the global optimum of the Sphere function.

This finding is not so strange, because the hypercube used for sampling is of fixed size throughout the optimization run, while the hyperball that holds the part of the search-space with improved fitness, decreases in size each time such an improvement is found. Thus we have proved a deficiency with localized sampling of the Sphere function using a fixed sampling-range. Namely that the closer it gets to the optimum, the less likely it is to find any improved positions. In other words, optimization methods that work by local sampling must decrease their search-range somehow.

References

[1] R. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[2] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67 – 82, 1997.

[3] D. Kincaid and W. Cheney. *Numerical Analysis.* Brooks/Cole, 2002.

[4] W.C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, 1991.

[5] C.G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6:222–231, 1970.

[6] R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13(3):317–322, 1970.

[7] D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[8] D.F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[9] J. Nocedal and S.J. Wright. *Numerical Optimization.* Springer-Verlag, 1999.

[10] C. Darwin. *On the Origin of Species (by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life).* 1859.

[11] J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

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

[13] T. Bäck, F. Hoffmeister, and H. Schwefel. A survey of evolution strategies. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 2–9, San Diego, California, 1991.

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

[15] J.R. Koza. *Genetic Programming : on the programming of computers by means of natural selection*. Bradford, MIT Press, 1992.

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

[17] R. Storn. On the usage of differential evolution for function optimization. In *Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS)*, pages 519–523, Berkeley, CA, USA, 1996.

[18] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, Perth, Australia, 1995.

[19] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. In *Proceedings of 1998 IEEE International Conference on Evolutionary Computation*, pages 69–73, Anchorage, AK, USA, 1998.

[20] J. Matyas. Random optimization. *Automation and Remote Control*, 26(2):246–253, 1965.

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

[22] Y. Shi and R.C. Eberhart. Parameter selection in particle swarm optimization. In *Proceedings of Evolutionary Programming VII (EP98)*, pages 591 – 600, 1998.

[23] R.C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. *Proceedings of the 2000 Congress on Evolutionary Computation*, 1:84 – 88, 2000.

[24] A. Carlisle and G. Dozier. An off-the-shelf PSO. In *Proceedings of the Particle Swarm Optimization Workshop*, pages 1 – 6, 2001.

[25] K. Price, R. Storn, and J. Lampinen. *Differential Evolution – A Practical Approach to Global Optimization*. Springer, 2005.

[26] J. Liu and J. Lampinen. On setting the control parameter of the differential evolution method. In *Proceedings of the 8th International Conference on Soft Computing (MENDEL)*, pages 11–18, Brno, Czech Republic, 2002.

[27] F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, Faculty of Natural and Agricultural Science, November 2001.

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

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

[30] D. Zaharie. Critical values for the control parameters of differential evolution algorithms. In *Proceedings of MENDEL 2002, 8th International Mendel Conference on Soft Computing*, pages 62–67, Bruno, 2002.

[31] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[32] M. Meissner, M. Schmuker, and G. Schneider. Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics*, 7(125), 2006.

[33] S.K. Smit and A.E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 399–406, Trondheim, Norway, 2009.

[34] T. Bäck. Self-adaptation in genetic algorithms. In *Proceedings of 1st European Conference on Artificial Life*, pages 263–271. MIT Press, 1992.

[35] A. Tuson and P. Ross. Co-evolution of operator settings in genetic algorithms. In *Proceedings of the Third AISB Workshop on Evolutionary Computing*, Berlin, 1996. Springer.

[36] J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft Computing*, 9(6):448–462, 2005.

[37] L.A. Zadeh. Fuzzy logic and approximate reasoning. *Synthese*, 30:407–428, 1975.

[38] A.K. Qin and P.N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Proceedings of the IEEE congress on evolutionary computation (CEC)*, pages 1785–1791, 2005.

[39] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark functions. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[40] K.E. Parsopoulos and M.N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1:235–306, 2002.

[41] H-P. Schwefel. Collective phenomena in evolutionary systems. In *Proceedings of the 31st Annual Meeting on Problems of Constancy and Change – The Complementarity of Systems Approaches to Complexity*, volume 2, pages 1025–1033, Budapest, 1987.

[42] Z. Tu and Y. Lu. A robust stochastic genetic algorithm (StGA) for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 8(5):456–470, 2004.

[43] Z. Tu and Y. Lu. Errata to "A robust stochastic genetic algorithm (StGA) for global numerical optimization". *IEEE Transactions on Evolutionary Computation, Accepted*, 2008.

[44] S. Hawking. *A Brief History of Time*. Bantam Press, 1988.

[45] S. Hawking. *The Universe in a Nutshell*. Bantam Press, 2001.

[46] P. Ball. *The Self-Made Tapestry: pattern formation in nature*. Oxford University Press, 2001.

[47] J.H. Holland. *Hidden Order: how adaptation builds complexity*. Addison Wesley Longman Publishing Co., Inc., 1995.

[48] J.H. Holland. *Emergence: from chaos to order*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[49] S. Johnson. *Emergence: the connected lives of ants, brains, cities, and software*. Scribner, 2002.

[50] D.R. Hofstadter. *Gödel, Escher, Bach: an eternal golden braid*. Basic Books, 1979.

[51] J. Kennedy. The particle swarm: social adaptation of knowledge. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 303–308, Indianapolis, USA, 1997.

[52] D. Bratton and T. Blackwell. A simplified recombinant PSO. *Journal of Artificial Evolution and Applications*, 2008. Article ID 654184.

[53] L. Pellarin. Learning and optimization of subjective problems – M.Sc.IT Thesis. *IT University of Copenhagen, Denmark*, 2007.

[54] R. Hooke and T.A. Jeeves. "Direct Search" solution of numerical and statistical problems. *Journal of the Association for Computing Machinery (ACM)*, 8(2):212–229, 1961.

[55] J. Kiefer. Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, 4(3):502–506, 1953.

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

[57] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, 1997.

[58] E.D. Dolan, R.M. Lewis, and V.J. Torczon. On the local convergence properties of pattern search. *SIAM Journal on Optimization*, 14(2):567–583, 2003.

[59] D.E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 2nd edition, 1998.

[60] L.A. Rastrigin. The convergence of the random search method in the extremal control of a many parameter system. *Automation and Remote Control*, 24(10):1337–1342, 1963.

[61] V.A. Mutseniyeks and L.A. Rastrigin. Extremal control of continuous multi-parameter systems by the method of random search. *Engineering Cybernetics*, 1:82–90, 1964.

[62] M.A. Schumer and K. Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13(3):270–276, 1968.

[63] J.P. Lawrence III and K. Steiglitz. Randomized pattern search. *IEEE Transactions on Computers*, C-21(4):382–385, 1972.

[64] G. Schrack and M. Choit. Optimized relative step size random searches. *Mathematical Programming*, 10(1):230–244, 1976.

[65] N. Baba. Convergence of a random optimization method for constrained optimization problems. *Journal of Optimization Theory and Applications*, 33(4):451–461, 1981.

[66] F.J. Solis and R.J-B. Wets. Minimization by random search techniques. *Mathematics of Operation Research*, 6(1):19–30, 1981.

[67] C.C.Y. Dorea. Expected number of steps of a random optimization method. *Journal of Optimization Theory and Applications*, 39(3):165–171, 1983.

[68] M.S. Sarma. On the convergence of the Baba and Dorea random optimization methods. *Journal of Optimization Theory and Applications*, 66(2):337–343, 1990.

[69] R. Luus and T.H.I. Jaakola. Optimization by direct search and systematic reduction of the size of search region. *American Institute of Chemical Engineers Journal (AIChE)*, 19(4):760–766, 1973.

[70] G.G. Nair. On the convergence of the LJ search method. *Journal of Optimization Theory and Applications*, 28(3):429–434, 1979.

[71] R. Luus. Use of line search in the Luus-Jaakola optimization procedure. In *Proceedings of the Third IASTED International Conference on Computational Intelligence*, pages 128–135, Alberta, Canada, 2007.

[72] J. Jezowski, R. Bochenek, and G. Ziomek. Random search optimization approach for highly multi-modal nonlinear problems. *Advances in Engineering Software*, 36(8):504–517, 2005.

[73] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087 – 1092, 1953.

[74] Z. Michalewicz and D.B. Fogel. *How To Solve It: modern heuristics*. Springer-Verlag, 2000.

[75] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671 – 680, 1983.

[76] M. Løvbjerg and T. Krink. The lifecycle model: combining particle swarm optimisation, genetic algorithms and hillclimbers. In *Proceedings of Parallel Problem Solving from Nature VII (PPSN)*, pages 621 – 630, 2002.

[77] S. Chalup and F. Maire. A study on hill climbing algorithms for neural network training. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 2014–2021. IEEE Press, 1999.

[78] R. Storn. Private correspondance, 2008.

[79] R.C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC)*, pages 84–88, San Diego, CA, USA, 2000.

[80] X. Yao, Y. Ling, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, 1999.

[81] H.H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Computer Journal*, 3:175–184, 1960.

[82] K.A. De Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[83] A. Törn and A. Zilinskas. *Global Optimization*. Springer-Verlag New York, Inc., 1989.

[84] D. Whitley, S.B. Rana, J. Dzubera, and K.E. Mathias. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1-2):245–276, 1996.

[85] T. Bäck. *Evolutionary Algorithms in Theory and Practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK, 1996.

[86] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical Report KanGAL 2005005, Nanyang Technological University, Singapore, and Kanpur Genetic Algorithms Laboratory, Kanpur, India, 2005.

[87] R.C. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 81 – 86, 2001.

[88] T. Bäck. Parallel optimization of evolutionary algorithms. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature (PPSN)*, pages 418–427, London, UK, 1994. Springer-Verlag.

[89] M. Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Belgium, 2004.

[90] V. Nannen and A.E. Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 183–190, Seattle, USA, 2006.

[91] E.K. Burke, M.R. Hyde, G. Kendall, and J. Woodward. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO)*, pages 1559–1565, London, England, 2007.

[92] A.S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):21–61, 2008.

[93] R.E. Mercer and J.R. Sampson. Adaptive search using a reproductive metaplan. *Kybernetes (The International Journal of Systems and Cybernetics)*, 7:215–228, 1978.

[94] A.J. Keane. Genetic algorithm optimization in multi-peak problems: studies in convergence and robustness. *Artificial Intelligence in Engineering*, 9:75–83, 1995.

[95] V. Nannen and A.E. Eiben. Efficient relevance estimation and value calibration of evolutionary algorithm parameters. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 103–110, Singapore, 2007.

[96] O. François and C. Lavergne. Design of evolutionary algorithms – a statistical perspective. *IEEE Transactions on Evolutionary Computation*, 5(2):129–148, 2001.

[97] A. Ben-Israel, A. Ben-Tal, and A. Charnes. Necessary and sufficient conditions for a pareto optimum in convex programming. *Econometrica*, 45(4):811–20, May 1977.

[98] J. Horn, N. Nafpliotis, and D.E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE International Conference on Evolutionary Computation*, volume 1, pages 82–87, New Jersey, USA, 1994.

[99] C.M. Fonseca and P.J. Fleming. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, Urbana-Champaign, IL, USA, 1993.

[100] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[101] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[102] A.W. Iorio and X. Li. Solving rotated multi-objective optimization problems using differential evolution. In *Australian Conference on Artificial Intelligence*, pages 861–872, 2004.

[103] T. Bäck, D.B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. IOP Publishing and Oxford University Press, 1997.

[104] R. Luus. Private correspondance, 2007.

[105] A.J. Chipperfield. Private correspondance, 2006–2008.

[106] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[107] T. Krink, B. Filipic, G.B. Fogel, and R. Thomsen. Noisy optimization problems - a particular challenge for differential evolution? In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 332–339, Portland, Oregon, 20-23 June 2004. IEEE Press.

[108] A. Carlisle and G. Dozier. Tracking changing extrema with adaptive particle swarm optimizer. In *Proceedings of the 5th Biannual World Automation Congress*, pages 265 – 270, 2002.

[109] A.N. Aizawa and B.W. Wah. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation*, 2(2):97–122, 1994.

[110] W.J. Gutjahr and G.C. Pflug. Simulated annealing for noisy cost functions. *Journal of Global Optimization*, 8(1):1–13, 1996.

[111] J.M. Fitzpatrick and J.J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3(2–3):101–120, 1988.

[112] M. Rattray and J. Shapiro. Noisy fitness evaluation in genetic algorithms and the dynamics of learning. In *Foundations of Genetic Algorithms 4*, pages 117–139. Morgan Kaufmann, San Francisco, CA, 1997.

[113] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 312–317, 1996.

[114] S. García, D. Molina, and M. Lozano. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC 2005 special session on real parameter optimization. *Journal of Heuristics*, 15(6):617–644, 2005.

[115] A. Auger and N. Hansen. Performance evaluation of an advanced local search evolutionary algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1777–1784, 2005.

[116] W. Huyer and A. Neumaier. SNOBFIT - stable noisy optimization by branch and fit. *ACM Transactions on Mathematical Software*, 35:Article 9, 2008.

[117] P. Kaelo and M. M. Ali. Differential evolution algorithms using hybrid mutation. *Computational Optimization and Applications*, 37(2):231–246, 2007.

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

[119] R. Storn. Differential evolution research – trends and open questions. In U. K. Chakraborty, editor, *Advances in Differential Evolution*, chapter 1. Springer, 2008.

[120] R. Storn. Optimization of wireless communications applications using differential evolution. In *SDR Technical Conference*, Denver, 2007.

[121] J. Brest, B. Bošković, S. Greiner, V. Žumer, and M.S. Maučec. Performance comparison of self-adaptive and adaptive differential evolution algorithms. *Soft Computing*, 11:617–629, 2007.

[122] G.T. Toussaint. Bibliography on estimation of misclassification. *IEEE Transactions on Information Theory*, 20(4):472–479, 1974.

[123] M.E.H. Pedersen and A.J. Chipperfield. Simplifying particle swarm optimization. *Applied Soft Computing*, 10:618–628, 2010.

[124] Z. Xinchao. A perturbed particle swarm algorithm for numerical optimization. *Applied Soft Computing*, 10:119–124, 2010.

[125] T. Niknam and B. Amiri. An efficient hybrid approach based on PSO, ACO and k-means for cluster analysis. *Applied Soft Computing*, 10:183–197, 2010.

[126] M. El-Abda, H. Hassan, M. Anisa, M.S. Kamela, and M. Elmasry. Discrete cooperative particle swarm optimization for FPGA placement. *Applied Soft Computing*, 10:284–295, 2010.

[127] M-R. Chena, X. Lia, X. Zhanga, and Y-Z. Lu. A novel particle swarm optimizer hybridized with extremal optimization. *Applied Soft Computing*, 10:367–373, 2010.

[128] P.W.M. Tsang, T.Y.F. Yuena, and W.C. Situ. Enhanced affine invariant matching of broken boundaries based on particle swarm optimization and the dynamic migrant principle. *Applied Soft Computing*, 10:432–438, 2010.

[129] C-C. Hsua, W-Y. Shiehb, and C-H. Gao. Digital redesign of uncertain interval systems based on extremal gain/phase margins via a hybrid particle swarm optimizer. *Applied Soft Computing*, 10:606–612, 2010.

[130] H. Liua, Z. Caia, and Y. Wang. Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. *Applied Soft Computing*, 10:629–640, 2010.

[131] K. Mahadevana and P.S. Kannan. Comprehensive learning particle swarm optimization for reactive power dispatch. *Applied Soft Computing*, 10:641–652, 2010.

[132] J. Riget and J.S. Vesterstrøm. A diversity-guided particle swarm optimizer – the ARPSO. *Technical Report 2002-02, Department of Computer Science, University of Aarhus*, 2002.

[133] M. Løvbjerg and T. Krink. Extending particle swarm optimisers with self-organized criticality. In *Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002)*, volume 2, pages 1588 – 1593, 2002.

[134] H-W. Ge, Y-C. Liang, and M. Marchese. A modified particle swarm optimization-based dynamic recurrent neural network for identifying and controlling nonlinear systems. *Computers and Structures*, 85(21-22):1611–1622, 2007.

[135] Z-H. Zhan, J. Zhang, Y. Li, and H.S-H. Chung. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 39:1362–1381, 2009.

[136] E. Ridge and D. Kudenko. Sequential experiment designs for screening and tuning parameters of stochastic heuristics. In *Proceedings of the Ninth International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 27–34, Reykjavik, Iceland, 2006.

[137] T. Bartz-Beielstein, K.E. Parsopoulos, and M.N. Vrahatis. Analysis of particle swarm optimization using computational statistics. In *Proceedings of International Conference of Numerical Analysis and Applied Mathematics (ICNAAM)*, pages 34–37, Chalkis, Greece, 2004.

[138] S. Bengio, Y. Bengio, and J. Cloutier. Use of genetic programming for the search of a new learning rule for neural networks. In *International Conference on Evolutionary Computation*, pages 324–327, 1994.

[139] A. Radi and R. Poli. Discovering efficient learning rules for feedforward neural networks using genetic programming. In *Recent Advances in Intelligent Paradigms and Applications*, pages 133–159. Physica-Verlag GmbH, 2003.

[140] R. Poli, W.B. Langdon, and O. Holland. Extending particle swarm optimisation via genetic programming. In *EuroGP Lecture Notes in Computer Science (LNCS)*, pages 291–300. Springer-Verlag, Berlin, 2005.

[141] C. Ryan, J.J. Collins, and M. O'Neill. Grammatical evolution: evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming (EuroGP)*, volume 1391 of *Lecture Notes in Computer Science*, pages 83–95, Berlin, Germany, 1998.

[142] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[143] M. O'Neill and C. Ryan. *Grammatical Evolution: evolutionary automatic programming in an arbitrary language*. Kluwer Academic Publishers, 2003.

[144] A. Ortega, M. Cruz, and M. Alfonseca. Christiansen grammatical evolution: grammatical evolution with semantics. *IEEE Transactions on Evolutionary Computation*, 11(1):77–90, 2007.

[145] M. O'Neill and C. Ryan. Grammatical evolution by grammatical evolution: the evolution of grammar and genetic code. In *Proceedings of the European Conference on Genetic Programming*, pages 138–149, Coimbra, Portugal, 2004.

[146] J. Schmidhuber. Gödel machines: fully self-referential optimal universal problem solvers. In *Artificial General Intelligence*, pages 201–228. Springer Verlag, 2006.

[147] J. Schmidhuber. Gödel machines: towards a technical justification of consciousness. In *Adaptive Agents and Multi-Agent Systems III (LNCS 3394)*, pages 1–23. Springer Verlag, 2005.

[148] E.W. Weisstein. *CRC Concise Encyclopedia of Mathematics*. Chapman & Hall/CRC, 2nd edition, 2002.