Particle Swarms for Constrained Optimization

Partikelschwärme für Optimierungsprobleme mit Nebenbedingungen

Der Technischen Fakultät der
Universität Erlangen-Nürnberg
zur Erlangung des Grades

D O K T O R - I N G E N I E U R

vorgelegt von

Sabine Helwig

Erlangen 2010

Als Dissertation genehmigt von der Technischen
Fakultät der Universität Erlangen-Nürnberg

Tag der Einreichung:	23. April 2010
Tag der Promotion:	19. Juli 2010
Dekan:	Prof. Dr.-Ing. Reinhard German
Berichterstatter:	Prof. Dr. rer. nat. Rolf Wanka
	Prof. Dr. rer. nat. Martin Middendorf

Acknowledgments

I would like to thank my supervisor Prof. Dr. Rolf Wanka for his support, guidance and encouragement throughout my graduate studies, the open atmosphere in his group, his availability to answer my questions, and his scientific and editorial advice.

I am very grateful to Frank Neumann, Stefan Wittmann, Tobias Stoll, Gabriella Kókai, Jürgen Branke, and Sanaz Mostaghim for technical discussions and successful cooperation.

Many thanks for implementations and interesting discussions go to all students whose semester and diploma theses I supervised or co-supervised: Johannes Jordan, Alexandr Gnezdilov, Thomas Ritscher, Matthias Hoffmann, Ludmila Omeltschuk, and Valerian Brem.

I am grateful to all collegues who made the chair of Hardware-Software-Co-Design a pleasant place to work, and in particular to Josef Angermeier and Moritz Mühlenthaler for proofreading parts of this thesis.

Finally, I would like to thank my friends and family for their support. I am particularly grateful to my brother Wolfram for thorough proofreading of some of my theoretical works, and, most of all, to Stefan for his support and patience throughout the last years.

Abstract

Particle swarm optimization (PSO) is an optimization approach from the field of artificial intelligence. A population of so-called particles moves through the parameter space defined by the optimization problem, searching for good solutions. Inspired by natural swarms, the movements of the swarm members depend on own experiences and on the experiences of adjacent particles.

PSO algorithms are mainly used for continuous optimization problems, whose feasible space is often restricted by a set of constraints. A special case are box constraints, which define upper and lower bounds for the problem parameters. In the literature, there exist several so-called bound handling strategies to integrate box constraints in PSO algorithms, such as setting infeasible particles to the nearest feasible position or reflecting them at the boundary.

In this thesis, various aspects of box-constrained particle swarm optimization are examined. The core of this work is the theoretical analysis of the impact of box constraints for particle swarm optimization. It is shown mathematically that initial particle swarm performance strongly depends on the chosen bound handling strategy due to the fact that, with overwhelming probability, many particles leave the feasible space at the beginning of the optimization. Moreover, by using a simplified PSO model, is shown that this effect can be reduced if particle velocities are scaled with respect to the problem dimensionality. A thorough experimental evaluation shows that bound handling also significantly influences the final solution quality of a particle swarm optimizer, especially when applied to high-dimensional problems. Three way to cope with these results in practical PSO applications are presented: The careful selection of bound handling strategies, the use of self-adaptation, and the use of velocity adaptation. Finally, the investigated PSO algorithms are applied to an optimization problem from the field of mechanical engineering.

Kurzfassung

Partikelschwarmoptimierung (PSO) ist ein Optimierungsverfahren aus dem Bereich der Künstlichen Intelligenz. Eine Population sogenannter Partikel bewegt sich auf der Suche nach guten Lösungen durch den durch das Optimierungsproblem festgelegten Suchraum. Inspiriert von natürlichen Schwärmen, hängen die Bewegungen der Schwarmmitglieder sowohl von eigenen Erfahrungen als auch von den Erfahrungen benachbarter Partikel ab.

PSO-Algorithmen werden vor allem für kontinuierliche Optimierungsprobleme eingesetzt, deren Gültigkeitsbereich oftmals durch eine Reihe von Nebenbedingungen beschränkt ist. Als Spezialfall hat der Gültigkeitsbereich die Form eines hochdimensionalen Quaders, d.h., für jeden Parameter des Optimierungsproblems sind obere und untere Schranken festgelegt. In der Literatur existieren zahlreiche Möglichkeiten, quaderförmige Suchraumbeschränkungen in PSO-Algorithmen zu integrieren. So können ungültige Partikel beispielsweise auf die Suchraumgrenze gesetzt oder an dieser reflektiert werden.

In dieser Arbeit wird das Verhalten von Partikelschwärmen in beschränkten Suchräumen untersucht. Kern der Arbeit ist die theoretische Analyse der Anwendung von PSO-Algorithmen auf Optimierungsprobleme mit quaderförmigen Suchraumbeschränkungen. Es wird bewiesen, dass viele Partikel den Gültigkeitsbereich zu Beginn der Optimierung mit überwältigender Wahrscheinlichkeit verlassen. Als Konsequenz ergibt sich, dass die Art und Weise, wie ungültige Partikel behandelt werden, großen Einfluss auf das anfängliche Schwarmverhalten hat. Unter Verwendung eines vereinfachten PSO-Modells wird weiterhin gezeigt, dass dieser Effekt reduziert werden kann, wenn die Partikelgeschwindigkeiten an die Suchraumdimensionalität angepasst werden. Ausführliche experimentelle Studien zeigen, dass, insbesondere bei Anwendung auf hochdimensionale Optimierungsprobleme, die Strategie der Behandlung ungültiger Partikel auch auf die finale Lösungsqualität signifikanten Einfluss hat. Es werden drei Möglichkeiten vorgestellt, mit diesen Resultaten in der Praxis umzugehen: Die sorgfältige Auswahl von Strategien zur Behandlung ungültiger Partikel, die Verwendung von Selbstadaption und der Einsatz von Geschwindigkeitsadaption. Abschließend werden die untersuchten PSO-Algorithmen auf ein Optimierungsproblem aus dem Bereich Maschinenbau angewendet.

Contents

1. Introduction

Artificial intelligence (AI) [RN03, Win92] is a branch of computer science, which is concerned with the design of intelligent entities, and includes a broad variety of topics such as knowledge representation, automated reasoning, learning, and perception. In *computational swarm intelligence (CSI)* [BDT99, Eng05, KE01], which is a subfield of AI, the focus has shifted from the design of individual intelligent entities to the design of intelligent swarm behavior. CSI is inspired by natural swarms, such as fish schools, bird flocks, and ant colonies, in which problem-solving behavior emerges from the interaction of (usually simple) individuals.

Particle swarm optimization (PSO) [KE95, EK95, KE01] is a computational swarm intelligence method for *global optimization*. The task of global optimization is the *minimization* or *maximization* of an objective function $f : \mathcal{S} \to \mathbb{R}$, where $\mathcal{S}$ is an arbitrary n-dimensional *search space*, e.g., $\mathcal{S} \subseteq \mathbb{R}^n$. Consider the following example, adapted from [RN03]: The locations of k new airports should be determined such that the sum of squared distances from each city of a specified set of cities to its nearest airport is minimized. The goal is to determine the coordinates

$$(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$$

of the k new airports. The complexity of the problem increases if further real world circumstances, e.g., accessibility, existing infrastructure, or other economic aspects, are taken into account. Standard mathematical approaches such as solving $\nabla f(\vec{x}) = \vec{0}$, which is a necessary condition for $\vec{x}$ being an optimal solution, are often not applicable in such contexts due to the lack of a functional representation of f. Moreover, even if a continuously differentiable representation of f is available, solving the system of equations $\nabla f(\vec{x}) = \vec{0}$ is usually algorithmically as difficult as solving the original optimization problem [Ber95].

PSO algorithms do not take the functional representation of the objective function f into account, and are therefore suited for *black box optimization*. A typical situation is depicted in Figure 1.1. Due to the fact that the application of PSO algorithms is not restricted to a specific kind of optimization problems, they belong to the class of *meta-heuristic* optimization approaches. Since their invention, PSO algorithms were successfully applied to various continuous and discrete optimization problems. Example PSO applications are listed in Table 1.1. Other popular meta-heuristic optimization approaches are *simulated annealing*, *genetic algorithms*, and *ant colony optimization*.

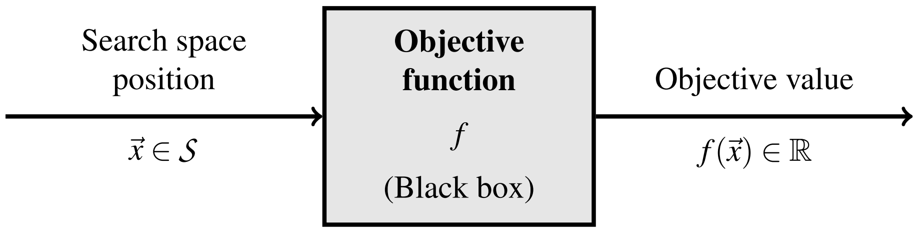


Figure 1.1: Black box optimization.

Particle swarm optimization is a stochastic optimization approach, which means that random numbers are involved when applying PSO to an optimization task. Theoretical analyses of PSO algorithms are very challenging due to the stochastic nature of particle swarm optimization and due to the patterns that emerge from the interactions of the participating individuals, which are hard to model and to analyze. Therefore, PSO algorithms are typically experimentally compared with each other and with other black box optimization algorithms by using well-defined *benchmark optimization problems*, such as the CEC 2005 benchmark suite [SHL+05]. Up to now, general theoretical results concerning the quality of the solutions obtained with PSO algorithms do not exist.

Many optimization tasks (e.g., [Abi02,HES03a,ZWLK09]) are subject to so-called *constraints*, which means that not an arbitrary solution to the problem is searched for, but a solution that complies with certain restrictions. For instance, in the above example, the airport coordinates have to be chosen such that the airports are located inside a specified country. A special case of constraints are so-called *box constraints*, which define upper and lower bounds for the problem parameters. Suppose that, in our airport example, not only the airport locations, but also the runway lengths have to be determined. As it does not make sense to construct a runway with negative length, the respective parameters are bounded to be greater than zero or to be greater than a specified minimum runway length. Most benchmarks that are used to test PSO algorithms include box constraints, e.g., most of the CEC 2005 benchmarks [SHL+05]. Optimization problems with box constraints are also denoted as *box-constrained optimization problems* in the following.

1.1 Contributions

In this thesis, various aspects of box-constrained particle swarm optimization are examined. In constrained optimization, solutions that do not comply with the defined constraints are called *infeasible*. In case of box constraints there exist various so-called *bound handling strategies* to cope with infeasible particles, such as moving them to the nearest feasible position.

In the following, it is shown that the performance of a particle swarm optimizer

Table 1.1: Example PSO applications

Application	References
Training of artificial neural networks	[KE01]
Optimization of electrical power systems	[YKF$^+$00, Abi02, MF02a]
Optimization of wireless communication systems	[ZWLK09]
Design of pressure vessels	[HES03a]
Biomedical image registration	[WSZ$^+$04]
Relative positioning	[SWHP07, GWHK09]
Scheduling	[AP09]
Traveling salesperson problem	[Cle00]
N-queens	[HES03b]

significantly depends on the chosen bound handling strategy when solving high-dimensional box-constrained optimization problems. The solutions obtained with particle swarm optimization can be considerably improved if an appropriate bound handling method is selected. Alternative methods to cope with this result, such as self-optimization and velocity adaptation are described. The results presented in this thesis provide a better understanding of the peculiarities of box-constrained optimization problems and support the application of PSO algorithms to such problems.

Contributions to Box-constrained Particle Swarm Optimization

The contributions to box-constrained particle swarm optimization can be structured into theoretical and experimental results, and were mostly published in [HW07,HW08, HNW09].

For the theoretical analysis, PSO is described as an iterative stochastic process, similar to the models used in the analyses of Jiang et al. [JLY07a,JLY07b] and Poli et al. [PB07,Pol08]. Based on this model, particle initialization and particle trajectories in the first iteration are studied. It is first shown that, when solving high-dimensional problems, particles are initialized very close to the search space boundary defined by the box constraints of the optimization problem. In a second step, particle trajectories in the first iteration are analyzed, considering three different particle velocity initialization strategies. It is proved that many particles become infeasible in the first iteration of a PSO algorithm, with overwhelming probability. The higher the search space dimensionality the more probable a particle will leave the initialization space. This theoretical result implies that bound handling has a very strong impact on initial particle swarm behavior.

Furthermore, the impact of a particle's velocity on its tendency to become infeasible is analyzed by using a simplified PSO model. It is shown mathematically that the probablity that a particle leaves the feasible space can be strongly reduced if the interval from which the velocities are chosen is scaled with respect to the problem dimensionality.

Finally, the well-known Sphere benchmark is considered as an example to demonstrate some of the peculiarities of high-dimensional box-constrained particle swarm optimization.

The experimental evaluation addresses the following issues in order to confirm and to extend the theoretical results, and to support practical particle swarm optimization:

Velocity initialization: In the theoretical analysis, three velocity initialization strategies are considered. It is shown that none of the strategies is able to prevent that many particles become infeasible at the beginning of the optimization process. The experimental comparison of these three strategies confirms that velocity initialization has only minor impact on the performance of a particle swarm optimizer.

Significance of bound handling: It was proved that bound handling has strong impact on initial particle swarm behavior. An experimental comparison shows that also the final solution quality is significantly affected by the chosen bound handling strategy. The higher the problem dimensionality, the more noticeable are the performance differences.

Strengths and weaknesses of selected strategies: Bound handling has strong impact on particle swarm performance and particle swarm behavior. A straightforward way to cope with this fact is to carefully select the bound handling strategy according to the specific application. In order to assist in this process, various commonly-used bound handling methods are experimentally compared. Strengths and weaknesses of the investigated strategies are discussed.

The experimental results were obtained by using standard benchmark problems and interpreted by means of statistical methods and tests.

Contributions to Adaptive Particle Swarm Optimization

Two adaptive particle swarm optimizers are presented: *Multi-Swarm PSO with Migration (MPSO)* [JHW08] and *PSO with Velocity Adaptation* [HNW09, HNW10]. Both algorithms are investigated experimentally and discussed with focus on box-constrained optimization.

The goal of Multi-Swarm PSO with Migration is to reduce the necessity of manual parameter adjustment for PSO applications. Instead, the parameters are dynamically adapted during the optimization by using a subswarm approach. The performance of MPSO and its ability to adapt the bound handling strategy to the current problem are investigated experimentally.

The second adaptive particle swarm optimizer, PSO with Velocity Adaptation, was derived from the theoretical results in order to reduce the importance of bound handling for particle swarm optimization. Experimentation shows that PSO with Velocity Adaptation is less sensitive to the chosen bound handling strategy than standard particle swarm optimization, and often provides superior results at the same time.

The use of these adaptive PSO algorithms is not restricted to box-constrained optimization problems.

Contributions to PSO Application

Finally, particle swarm optimization is applied to the relative positioning problem, which often has to be solved in the field of tolerance analysis in mechanical engineering. The optimization task is a six-dimensional box-constrained problem, investigated in cooperation with the Chair of Engineering Design (Department of Mechanical Engineering) of the University of Erlangen-Nuremberg (see also [SWHP07, GWHK09]). The impact of bound handling on this rather low-dimensional problem is analyzed. Both adaptive algorithms investigated in this thesis are applied to relative positioning, and their performance is compared with standard particle swarm optimization.

1.2 Overview

This thesis is structured as follows: In Chapter 2, the concept of particle swarm optimization is introduced, and related work is presented. First, PSO algorithms for continuous, binary, and combinatorial optimization are described, and PSO parameters and recommended settings are discussed. Section 2.2 summarizes various kinds of particle cooperation. Particle swarm optimization for constrained optimization is detailed in Section 2.3. After presenting some general constraint handling techniques for stochastic search algorithms, available bound handling methods for box-constrained particle swarm optimization are categorized and discussed. Previous theoretical results in the field of particle swarm optimization are presented in Section 2.4. These studies mostly concentrate on the parameters of the PSO movement equations, but do neither take high-dimensional parameter spaces nor constraints into account. A brief overview on multi-objective particle swarm optimization is given afterwards, before concluding Chapter 2 with a short description of related meta-heuristic optimization approaches.

Chapter 3 covers the theoretical analysis of box-constrained particle swarm optimization. After introducing the examined PSO model in Section 3.1, particle initialization is analyzed in Section 3.2. The particles' behavior at the beginning of the optimization is analyzed theoretically in Section 3.3, taking three velocity initialization strategies into account. Afterwards, the impact of particle velocities on their tendency to become infeasible is studied in Section 3.4. For this investigation, a simplified PSO model is used. In Section 3.5, some of the pecularities of high-dimensional particle swarm optimization are demonstrated by using the well-known Sphere benchmark as an example. The implications of the theoretical results for practical PSO application are finally discussed in Section 3.6.

Chapter 4 contains the experimental contributions to box-constrained PSO. The experimental procedure is clarified in Section 4.1. In the same section, the statistical tools used in the experimental evaluation are briefly described. The investigated test problems are summarized in Section 4.2. Afterwards, experimental results are presented and discussed. First, velocity initialization is studied in Section 4.3. Various bound handling methods are analyzed in detail in Section 4.4. After demonstrating the significance of bound handling for particle swarm optimization, the strengths and weaknesses of several commonly-used methods are discussed on the basis of the experimental results.

Chapter 5 is devoted to the contributions in the field of adaptive particle swarm optimization. First, existing adaptation strategies are categorized and presented in Section 5.1. Afterwards, *Multi-Swarm PSO with Migration* and *PSO with Velocity Adaptation* are presented in Section 5.2 and Section 5.3, respectively. Both algorithms are analyzed experimentally, with focus on their benefits for box-constrained particle swarm optimization.

Finally, the application of particle swarm optimization in the field of mechanical engineering is presented in Chapter 6. A brief introduction into the field of tolerance analysis is given in Section 6.1. Relative positioning is formulated as a continuous optimization problem afterwards in Section 6.2. Experimental results are presented in Section 6.3.

2. Particle Swarm Optimization (PSO)

Particle swarm optimization (PSO) is a nature-inspired algorithm for *global optimization*. The task of global optimization is to minimize or to maximize an objective function $f : \mathcal{S} \to \mathbb{R}$. In this thesis, minimization problems are assumed, which means that the goal is to find a solution $x^* \in \mathcal{S}$ such that

$$\forall x \in \mathcal{S} : f(x^*) \leq f(x) \ \ .$$

A solution x^* that satisfies this condition is called a *global minimum*. If there exists an $\varepsilon > 0$ such that

$$\forall x \text{ with } ||x - x^*|| < \varepsilon : f(x^*) \leq f(x) \ \ ,$$

the solution x^* is called *local minimum*.

This chapter provides an introduction into particle swarm optimization methods. After presenting background information on flocks, herds, and schools, PSO algorithms for continuous, binary and combinatorial optimization problems are described in the following section.

2.1 The PSO Algorithm

Particle swarm optimization was first presented in 1995 by Kennedy and Eberhart [KE95, EK95]. A detailed description with a lot of background information can be found in their textbook *Swarm Intelligence* [KE01]. The algorithm is inspired by the social interaction of individuals living together in groups, e.g., bird flocks, fish schools, or human societies.

2.1.1 Flocks, Herds, and Schools

The synchronous movement of birds inside a flock or land animals in their herd was analyzed in the 1980s by both computer scientists and zoologists [Rey87, HG90]. While the aim of computer scientists was the realistic visualization of bird flocks, fish schools, or herds of land animals, zoologists were interested in the dynamics of these natural systems. In 1987, Reynolds [Rey87] simulated the flight of birds by assigning a simple set of rules to each single bird. His algorithm is decentralized, deterministic, and each individual's perception is restricted to its local neighborhood. The state of a simulated bird consists of its geometrical shape model, orientation, position in the respective coordinate system and a so-called *velocity*, which is the combination of both speed and direction of the bird's movement. The following rules, listed in decreasing order of importance, were assigned to each individual:

- *Collision Avoidance:* Avoid collisions with other flock members.
- *Velocity Matching:* Try to fly in same direction and with same speed than nearby flockmates.
- *Flock Centering:* Move to the center of nearby flock members. Flock centering helps the simulated birds to stay together.

Using these simple rules and some additional restrictions like a maximum velocity and a maximum acceleration per individual, flock-like behavior was simulated. Complex movement patterns were the result of the interaction of simple individuals. The model can not only be used for the visualization of bird flocks, but also for fish schools and herds of land animals.

Reynold's approach is similar to the *particle systems* presented by Reeves [Ree83] in 1983. Reeves claimed that the motion of fuzzy objects like fire, smoke, clouds, or water, which do not have a well-defined, smooth surface, cannot be described by simple affine transformations commonly used in computer graphics. Instead, he modelled such systems by using thousands or even millions of so-called *particles* with each one having its own behavior. A particle is a point in three-dimensional space, having a position, velocity (again, velocity is the combination of speed and direction), color, transparancy, and lifetime. For each frame, each particle's velocity is calculated according the the system's characteristics, and added to its position. As we will see later, PSO has a very similar structure, and also the term particle can partly be traced back to Reeveses particle systems [KE95].

Similar to Reynold's approach, Heppner and Grenander [HG90] described the synchronous movements of birds by assigning a set of rules to each bird. However, their simulated birds were additionally attracted by a roost. In early simulations, which finally led to the PSO algorithm, Kennedy and Eberhart [KE95] extended this model: instead of knowing the exact position of the roost (or food source), their birds, also called *agents*, were able to evaluate the distance to it, the so-called *cornfield vector*. The simulated birds are then indirectly attracted by the roost (or food source). Through interaction with other flock members, each agent tries to minimize its cornfield vector, until it eventually arrives at its destination.

Initially, one of the goals of Kennedy's and Eberhart's studies was to derive a simplified model of human social behavior, in order to simulate social processes [KE95]. Hence, the interaction of the agents was inspired by findings in the field of social psychology and by socio-psychological models of human behavior. While birds move in three-dimensional space and try to avoid collisions, beliefs, thoughts and attitudes of human beings are points in an high-dimensional cognitive space, where collisions may occur. People who live in the same social group tend to become more and more similar, and norms and cultures emerge. The model which underlies PSO assumes that each individual has three main characteristics [KE01]: It *evaluates* stimuli of the environment, it *compares* itself with other members of its social group, and it *imitates* other individuals, preferably better ones. By using these three principles, each individual is able to learn from others, and to improve (optimize) itself to a certain degree.

2.1.2 PSO for Continuous Problems

The PSO algorithm [KE95, EK95, KE01, BK07] considers two main sources of influence for social learning processes: Individuals rely on their own previous experiences (*cognitive component*), and they imitate better group members (*social component*). Transforming these observations in an iteration-based optimization algorithm, a population of m individuals, which are from now on called *particles*, explores the n-dimensional search space $\mathcal{S}$ of an optimization problem with objective function $f : \mathcal{S} \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$. Without loss of generality, a minimization problem is assumed. Each particle i has a *position* $\vec{x}_{i,t}$ (where t is the iteration counter), a *fitness value* $f(\vec{x}_{i,t})$, and moves through the search space with a *velocity* $\vec{v}_{i,t}$. A position $\vec{z}_1 \in \mathcal{S}$ is called *better* than $\vec{z}_2 \in \mathcal{S}$ iff $f(\vec{z}_1) < f(\vec{z}_2)$. The best search space position particle i has visited until iteration t is its *private guide* $\vec{p}_{i,t}$. To each particle, a subset of all particles is assigned as its neighborhood (see Section 2.2 for more information about possible neighborhood structures). The best private guide of all neighbors of particle i is called its *local guide* $\vec{l}_{i,t}$. Besides the cognitive and the social component, and based on the model of bird flocks or fish schools, each particle additionally keeps a fraction of its old velocity, which results in the following update equations for particle swarm optimization:

$$\vec{v}_{i,t} = \omega \cdot \vec{v}_{i,t-1} + \underbrace{c_1 \cdot \vec{r}_{1,i,t} \odot (\vec{p}_{i,t-1} - \vec{x}_{i,t-1})}_{\text{cognitive component}} + \underbrace{c_2 \cdot \vec{r}_{2,i,t} \odot (\vec{l}_{i,t-1} - \vec{x}_{i,t-1})}_{\text{social component}} \tag{2.1}$$

$$\vec{x}_{i,t} = \vec{x}_{i,t-1} + \vec{v}_{i,t} \tag{2.2}$$

where ω, c_1, and c_2 are prespecified parameters, $\vec{r}_{1,i,t}$ and $\vec{r}_{2,i,t}$ are vectors of random real numbers whose components are drawn uniformly at random in $[0,1]$, and $\odot$ denotes element-by-element vector multiplication.

After each particle has computed its new position and velocity, the private guides of all particles are updated. Typically, a particle's private guide $\vec{p}_{i,t}$ is updated to its current position $\vec{x}_{i,t}$ iff $f(\vec{x}_{i,t}) < f(\vec{p}_{i,t-1})$ holds. However, other definitions are pos-

Algorithm 2.1 Particle swarm optimization

```
Require: Objective function $f : \mathcal{S} \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, PSO parameters
 1: for each particle $i$ ($i = 1, \dots, m$) do {Particle initialization}
 2:     Initialize position $\vec{x}_{i,0}$ and velocity $\vec{v}_{i,0}$ according to initialization strategy
 3:     Initialize private guide: $\vec{p}_{i,0} \leftarrow \vec{x}_{i,0}$
 4: end for
 5: Initialize neighborhood structure
 6: $t \leftarrow 0$
 7: repeat
 8:     $t \leftarrow t + 1$
 9:     for each particle $i$ ($i = 1, \dots, m$) do {Particle movement}
10:         Velocity update according to Equation (2.1)
11:         Position update according to Equation (2.2)
12:         if $\vec{x}_{i,t} \notin \mathcal{S}$ then
13:             Apply bound handling procedure
14:         end if
15:     end for
16:     for each particle i ($i = 1, \dots, m$) do {Private guide update}
17:         if success($\vec{x}_{i,t}, \vec{p}_{i,t-1}$) then
18:             $\vec{p}_{i,t} \leftarrow \vec{x}_{i,t}$
19:         end if
20:     end for
21: until termination criterion met
```

sible (see Section 2.1.5). For the update of the private guides, the objective function f has to be evaluated once for each particle in each iteration.

The PSO algorithm is given in Algorithm 2.1. It terminates as soon as a specified termination criterion is met, for instance, as soon as the best found solution was not improved during a certain amount of time or the iteration counter t exceeds a specified limit.

When applying particle swarm optimization to a given problem, the parameters of the algorithm, e.g., ω, c_1, c_2, and the neighborhood structure, have to be selected appropriately. Some parameter setting guidelines, which were previously extracted from theoretical studies and experimentation with commonly-used benchmark problems, are summarized in Section 2.1.5.

Particle swarm optimization was originally introduced for continuous optimization problems. However, there exist PSO variants for binary and combinatorial problems [KE97, Cle00], which are briefly described in Sections 2.1.3 and 2.1.4. Afterwards, the parameters of the PSO algorithm and initialization issues are discussed.

2.1.3 PSO for Binary Problems

In 1997, Kennedy and Eberhart [KE97] proposed a particle swarm optimizer for the optimization of pseudo-Boolean functions $f : \{0,1\}^n \to \mathbb{R}$. In their approach, each particle i has a position $\vec{x}_{i,t} \in \{0,1\}^n$, which is a binary vector, and a continuous velocity $\vec{v}_{i,t} \in [-V_{max}, V_{max}]^n$. Additionally, each particle stores its private guide $\vec{p}_{i,t}$ as a binary vector, and communicates with its neighbors to obtain the local guide $\vec{l}_{i,t}$. The PSO equation for the velocity update remains unchanged, despite the fact that no inertia weight ω is used:

$$\vec{v}_{i,t} = \vec{v}_{i,t-1} + c_1 \cdot \vec{r}_{1,i,t} \odot (\vec{p}_{i,t-1} - \vec{x}_{i,t-1}) + c_2 \cdot \vec{r}_{2,i,t} \odot (\vec{l}_{i,t-1} - \vec{x}_{i,t-1})$$

For all $d \in \{1, \ldots, n\}$, the d-th component of the velocity vector, $v_{i,t,d}$, is then mapped to the interval $[0,1]$ by using a mapping function $g : [-V_{max}, V_{max}] \to [0,1]$. The function value $g(v_{i,t,d})$ defines the probability that the respective component of the position vector is set to 1. Hence, a particle's position is updated according to:

```
if $R_{i,t,d} < g(v_{i,t,d})$ then
  $x_{i,t,d} \leftarrow 1$
else
  $x_{i,t,d} \leftarrow 0$
end if
```

where $R_{i,t,d}$ is drawn uniformly at random from [0,1]. As mapping function, an arbitrary function $g : [-V_{max}, V_{max}] \to [0,1]$ can be used, e.g., the monotonic increasing sigmoid function s, which is depicted in Figure 2.1:

$$s(v) = \frac{1}{1 + e^{-v}}$$

When using the sigmoid function, V_{max} may for instance be set to $V_{max} = 6$ [KE97] or to $V_{max} = 4$ [KE01]. As $s(-6) \approx 0.9975$ and $s(6) \approx 0.0025$ (respectively, $s(-4) \approx 0.9820$ and $s(4) \approx 0.0180$), for each bit in a particle's position vector there is a small probability that it is flipped. Hence, V_{max} is similar to a mutation rate in evolutionary algorithms. The smaller V_{max} is chosen, the higher is the probability that a bit is flipped. Based on their runtime analysis of this binary PSO, Sudholt and Witt [SW08] propose to scale V_{max} with the problem size and to set it to $V_{max} = \ln(n-1)$.

2.1.4 PSO for Combinatorial Problems

In order to design a particle swarm optimizer for combinatorial problems, Clerc [Cle00] first identified the main components of the PSO algorithm, and then redefined them with respect to the particular problem. In his general framework, the goal is to optimize an objective function $f : \mathcal{S} \to \mathcal{Y}$, where $\mathcal{S}$ is an arbitrary search space,

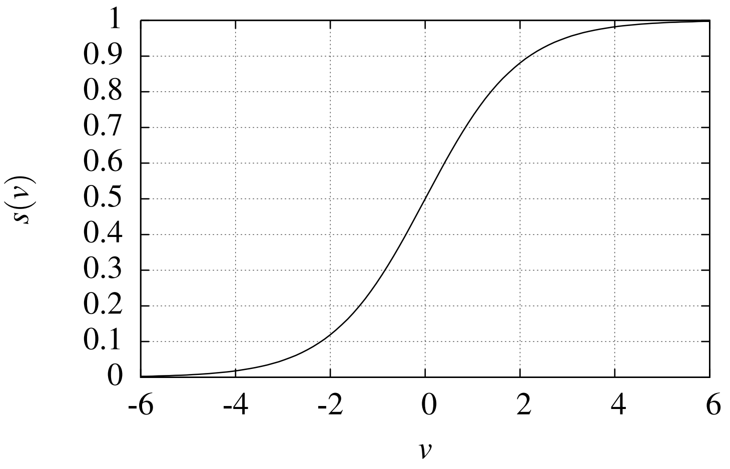


Figure 2.1: The sigmoid function $s(v) = \frac{1}{1+e^{-v}}$, which can be used as mapping function in the binary PSO.

and $\mathcal{Y}$ is an arbitrary, totally ordered objective space. Both $\mathcal{S}$ and $\mathcal{Y}$ may be finite sets.

In order to develop a PSO algorithm for combinatorial problems, the following PSO components must be redefined:

- A particle's position, representing a solution of the given problem
- A particle's velocity, representing the difference of two positions
- Subtraction: $position - position = velocity$
- Addition: $velocity + velocity = velocity$
- External multiplication: $r \cdot velocity = velocity$ with $r \in \mathbb{R}$
- Move: $position + velocity = position$

Note that, in contrast to the standard PSO formulas presented in Eq. (2.1) and Eq. (2.2), a single value is used instead of a vector of random real numbers for external multiplication.

Clerc illustrated this general PSO framework by designing a particle swarm optimizer for the *traveling salesperson problem (TSP)*. TSP is the problem of finding a Hamiltonian cycle[1] of minimum length in a complete weighted graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ and E denote the set of vertices and edges, respectively. A solution of a traveling salesperson problem, and hence, a particle's position, can be

[1]A Hamiltonian cycle in a graph G is a closed path which visits every node exactly once.

given as a permutation indicating in which order the nodes are visited. Let $w_{i,j}$ be the weight of edge $(v_i, v_j) \in E$, and $x_{i,t}$ be the position of particle i at time step t, with $x_{i,t} = (p_1, p_2, \dots p_n)$, $p_j \in V$ and $\forall j,k$ with $1 \leq j < k \leq n : p_j \neq p_k$. The objective value of $x_{i,t}$ is the length of the corresponding Hamiltonian cycle:

$$f(x_{i,t}) = w_{p_n,p_1} + \sum_{k=1}^{n-1} w_{p_k,p_{k+1}}$$

In the following, Clerc's definitions for the remaining components are briefly presented. A *velocity* is a list of transpositions. The *move* operator simply applies a list of transpositions (velocity) on a permutation (position). The *subtraction* of two positions x_1 and x_2, $x_2 - x_1$, yields a velocity vector v such that $x_1 + v = x_2$. Two velocities are added by appending the transpositions of the second velocity to those of the first velocity, possibly followed by a reduction of the length of the list. Note that this operation is not commutative. Finally, *external multiplication* truncates or expands a list of transpositions according to a specified scheme [Cle00].

Anghinolfi and Paolucci [AP09] proposed a discrete PSO algorithm for scheduling problems, which was developed on the basis of Clerc's general framework.

2.1.5 Parameters and Initialization

After having presented PSO algorithms for continuous, binary, and combinatorial optimization, the algorithmic parameters of particle swarm optimization are now discussed.

Inertia Weight and Acceleration Coefficients

The *inertia weight* ω was first introduced by Shi and Eberhart in 1998 [SE98]. Mostly, values smaller than 1 are used in order to reduce the swarm's exploration behavior over time. The so-called *acceleration coefficients* or *control parameters* c_1 and c_2 determine the relative influence of cognitive and social component for a particle's movement. Based on a convergence analysis of Clerc and Kennedy [CK02], the following relation between c_1, c_2, and ω was established, which is widely-used in particle swarm optimization (for details see also Section 2.4) [Cle06c]:

$$c_1 = c_2 = \frac{(\omega + 1)^2}{2} \tag{2.3}$$

with $2c_1/\omega = 2c_2/\omega > 4$. In the standard PSO presented by Bratton and Kennedy [BK07], the inertia weight is set to $\omega = 0.72984$, which results in $c_1 = c_2 \approx 1.49617$.

Another approach to select inertia weight and acceleration coefficients is to choose them randomly in specified intervals whenever a particle's velocity is updated [PC04].

Population Size

The population size m is often set to values between 20 and 50, and should be chosen according to the problem characteristics and dimensionality [PKB07]. Bratton and Kennedy [BK07] carried out experiments with different swarm sizes between 20 and 100 and reported that none of the swarm sizes performed clearly better or worse than the others on the tested benchmarks. Hence, it seems that the population size only slightly influences particle swarm performance.

In some implementations, the population size is adapted to the dimensionality n of the problem under consideration, and set to $m = 10 + 2\sqrt{2n}$ [Cle06a].

Velocity Clamping

In early PSO implementations (particularly in those without inertia weight, i.e., $\omega = 1$), the particles' velocities were component-wise limited to specified values in order to prevent that the magnitudes of the particles' positions and velocities rapidly increase [BK07]. Let $\mathcal{V} = [-V_{max,1}, V_{max,1}] \times [-V_{max,2}, V_{max,2}] \times \cdots \times [-V_{max,n}, V_{max,n}]$. When using velocity clamping, the standard velocity update equation Eq. (2.1) is followed by (see, e.g., [KE01]):

```
if $v_{i,t,d} > V_{max,d}$ then
    $v_{i,t,d} = V_{max,d}$
else if $v_{i,t,d} < -V_{max,d}$ then
    $v_{i,t,d} = -V_{max,d}$
end if
```

Based on a deterministic PSO model, Clerc and Kennedy [CK02] showed that a restriction of the particles' velocities is not necessary to obtain a convergent particle swarm. Nevertheless, the use of velocity clamping can significantly improve the performance of a PSO algorithm [ES00]. A detailed discussion on velocity clamping, including time-dependent and adaptive settings for $V_{max,d}$, was presented by Engelbrecht [Eng05, p. 109ff.].

Private Guide Update and Local Guide Selection

A particle's private guide $\vec{p}_{i,t}$ is defined as the best position it has visited so far, and is usually updated to a particle's current position $\vec{x}_{i,t}$ iff $f(\vec{x}_{i,t}) < f(\vec{p}_{i,t-1})$ [EK95]. However, if there are large regions of equal fitness, this update procedure might reduce exploration, as discussed in a similar context by Owen and Harvey [OH07]. Based on this reasoning, the following private guide update procedures arise:

- *Standard*, as defined above, and described by Eberhart and Kennedy [EK95]:

```
if $f(\vec{x}_{i,t}) < f(\vec{p}_{i,t-1})$ then
  $\vec{p}_{i,t} = \vec{x}_{i,t}$
else
  $\vec{p}_{i,t} = \vec{p}_{i,t-1}$
end if
```

- *Newest*, based on the discussion of Owen and Harvey [OH07]:

```
if $f(\vec{x}_{i,t}) \leq f(\vec{p}_{i,t-1})$ then
  $\vec{p}_{i,t} = \vec{x}_{i,t}$
else
  $\vec{p}_{i,t} = \vec{p}_{i,t-1}$
end if
```

- *Random*, used in [HNW09] and throughout this thesis, where R is distributed uniformly at random in $[0,1]$, and independently drawn at each occurence:

```
if $f(\vec{x}_{i,t}) < f(\vec{p}_{i,t-1})$ then
  $\vec{p}_{i,t} = \vec{x}_{i,t}$
else if $f(\vec{x}_{i,t}) > f(\vec{p}_{i,t-1})$ then
  $\vec{p}_{i,t} = \vec{p}_{i,t-1}$
else
  if $R < 0.5$ then
    $\vec{p}_{i,t} = \vec{x}_{i,t}$
  else
    $\vec{p}_{i,t} = \vec{p}_{i,t-1}$
  end if
end if
```

Similarly, if a particle has more than one best neighbor, its local guide is drawn uniformly at random among the candidates.

Initialization of Particle Positions

Particle positions are often initialized uniformly at random in the search space $\mathcal{S}$ [Cle06b, Eng05]. If $\mathcal{S}$ is unbounded, i.e., $\mathcal{S} = \mathbb{R}^n$ for a certain n, appropriate initialization ranges have to be found.

Some benchmark problems that are widely-used for the comparison of stochastic search algorithms, such as Sphere, Rosenbrock, and Rastrigin (function descriptions can be found in Chapter 4), have their global optimum at the center of the search space $\mathcal{S}$. Therefore, when performing benchmark analyses, individuals are often initialized in an asymmetric subspace of $\mathcal{S}$ in order to avoid that the performance of center-biased algorithms is overestimated. The initialization space of the Sphere benchmark can for instance be set to $[50,100]^n$ whereas the parameter space is $[-100,100]^n$ [BK07]. However, there is no reason to define asymmetric initialization ranges when analyzing algorithms by means of more modern benchmark suites, such as the CEC 2005 benchmarks [SHL+05], or when applying PSO to real world problems.

Initialization of Particle Velocities

The particles' velocities can be initialized to one of the following schemes:

- *Uniform*: Let $\mathcal{S} \subseteq \mathbb{R}^n$ be the search space of the given optimization problem. Particle velocities are drawn uniformly at random in a specified n-dimensional space. For instance, the d-th component of a particle's velocity vector is initialized uniformly at random in $[-V_{max,d}, V_{max,d}]$ [KE01], or uniformly at random in $[-(ub_d - lb_d)/2, (ub_d - lb_d)/2]$, if $\mathcal{S} = [ub_1, lb_1] \times [ub_2, lb_2] \times \cdots \times [ub_n, lb_n] \subseteq \mathbb{R}^n$ is rectangularly bounded [Cle06b].

- *Zero*: Velocities are initialized to zero [Eng05], i.e., $\vec{v}_{i,0} = \vec{0}$ for each particle i.

- *Half-diff*: The initial velocities are set to [C+07]

  $$\vec{v}_{i,0} = \tfrac{1}{2}(\vec{z}_i - \vec{x}_{i,0})$$

  where $\vec{x}_{i,0}$ is the initial position of particle i, and the vectors $\vec{z}_i$ are independently drawn uniformly at random in $\mathcal{S}$ for $i = 1,\ldots,m$. The *half-diff* strategy is illustrated in Figure 2.2.

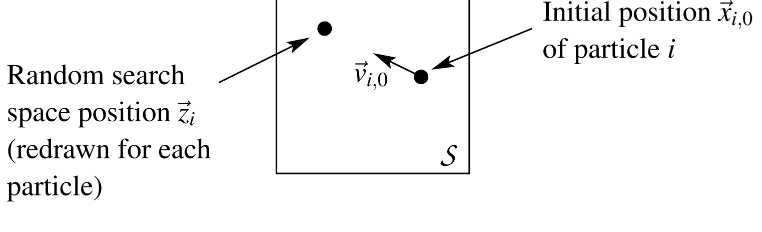

Initial position $\vec{x}_{i,0}$ of particle i
Random search space position $\vec{z}_i$ (redrawn for each particle)
$\vec{v}_{i,0}$
$\mathcal{S}$

Figure 2.2: *Half-diff* velocity initialization: $\vec{v}_{i,0} = \frac{1}{2}(\vec{z}_i - \vec{x}_{i,0})$

2.2 Social Interaction in PSO

Particle swarm optimization is inspired by models of social learning processes. When moving through the search space, each individual takes own experiences as well as those of neighboring particles into account. The social relationship among particles, the so-called *neighborhood topology*, is often visualized as directed or undirected graph in which nodes represent the particles, and edges specify which particles interact with each other, i.e., exchange their private guides.

In most PSO algorithms, the neighborhood relations are defined completely independent from the search space positions of the particles. Instead, neighborhood is solely based on the particles' identifiers. Communication is possible even if the search space distance is enormous. In very early simulations, and based on the model of bird flocking and fish schools, the so-called *Euclidean neighborhood* was used, which determines the neighbors of a particle according to distance meassures in the search space. This model was soon replaced due to its computational costs and often worse performance [BK07, PKB07]. However, for the computation of the private guide, information of nearby particles can be taken into account [Ken00]. This PSO variant was proposed by Kennedy and is presented at the end of this section.

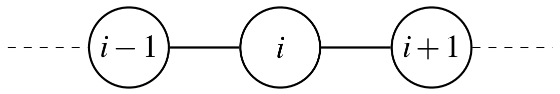
2.2.1 Static Neighborhood Topologies

A static neighborhood topology is usually represented as a graph, and used throughout the optimization run. Examples of static topologies are [EK95, KM02]:

- The fully connected graph, or *gbest* (*global best*) topology: All particles are connected.
- The *lbest(k)* (*local best*) topology, for an even $k \geq 0$: Each particle has exactly $k/2$ neighbors on each side. E.g., for $k = 2$, particle i is connected with particles $i-1$ and $i+1$ (the array is wrapped so that particles 1 and m are connected). For $k = 4$, particle i is additionally adjacent to particles $i-2$ and $i+2$, and so on. The special case *lbest(2)* is also called *ring* topology.
- The *grid* or so-called *von Neumann* topology: The particles are arranged like the nodes of a grid, and each particle has exactly four neighbors: the particle above, below, on the left hand side, and on the right hand side (using wraparound edges).

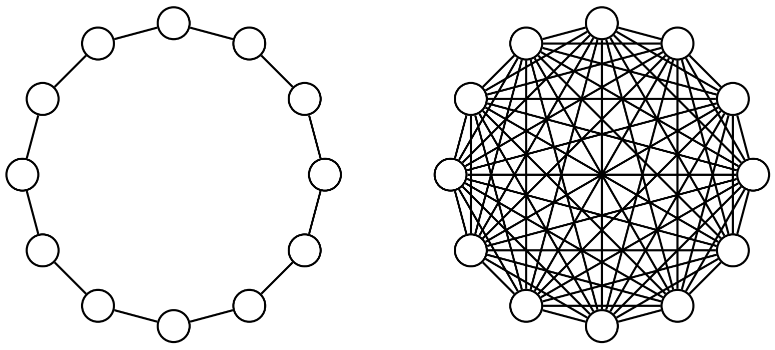
The *gbest*, *lbest(2)*, and *grid* topology are depicted in Figures 2.3 and 2.4. A particle can be included in or excluded from its own neighborhood.

The more neighbors an individual has, the faster its private guide is propagated among the particles. In a fully connected swarm, all particles use the same local guide for the velocity update, and the swarm often converges very fast. In contrast, the information flow in a sparsely connected graph can be very slow, as each particle solely provides its private guide to its neighbors, and not the information it has obtained from its neighbors. Assume, for instance, the ring topology. Then, particle i is connected with particles $i-1$ and $i+1$:



For all population sizes $m > 3$, particles $i-1$ and $i+1$ are not adjacent. If particle $i-1$ updates its private guide to its current position, it will be passed to particle i in the next iteration step. However, particle $i+1$ will not be informed of the promising search space region until particle i itself has improved its private guide.

The application of the ring topology was introduced in 1995 by Eberhart and Kennedy [EK95]. The authors reported that a particle swarm optimizer using the ring topology usually converged slower than a fully connected swarm, but is on the other hand more resistant to local optima. However, if the convergence speed of the ring topology is too slow for a particular application, the grid topology might be a good compromise of the fast converging fully connected graph and the ring topology: Kennedy and Mendes [KM02] compared more than 1000 neighborhood graphs on widely-used benchmark problems and concluded their study by recommending the grid topology due to its constantly good performance compared to other communication structures.



(a) Ring or lbest(2)

(b) Fully connected

Figure 2.3: Two commonly-used static neighborhood graphs for particle swarm optimization: The *ring* (or *lbest(2)*) topology and the fully connected (*gbest*) swarm.

2.2.2 Dynamic Neighborhood Topologies

Instead of using a predefined static topology, communication links may be modified during runtime. Some approaches dynamically adjust the neighborhood topology to

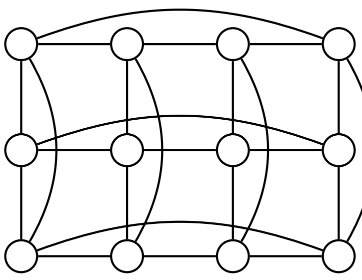


Figure 2.4: The *grid* (or *von Neumann*) topology.

the optimization problem to be solved [Sug99, RV03, Cle03]. These adaptive PSO variants are discussed in Chapter 5. The dynamic neighborhood topologies presented in this section do not take particle swarm performance into account.

In Liang's and Suganthan's [LS05] *Dynamic Multi-Swarm PSO*, the population is split into several small-sized and fully-connected subswarms which simultaneously explore different search space regions. The swarm is periodically regrouped into new subpopulations.

In a PSO variant presented by Mohais, Ward and Posthoff [MWP04], a directed neighborhood topology with gradual edge modifications is used. The communication structure is initialized at random. At the end of each iteration, a random edge is deleted from the neighborhood graph, and a new communication link with the same target is inserted. Alternatively, the communication structure can be completely reinitialized after a certain number of iterations, by keeping the out-degree of the neighborhood graph fixed throughout the optimization run [MMWP05]. Based on experimental investigations, Mohais et al. assigned 5 neighbors to each particle.

Clerc [Cle07] proposes to randomly reinitialize the neighborhood graph after either each iteration in which the best known solution was not improved or, inspired by a rumour spread model, after $L/2$ iterations, where L is the total number of communication links. In contrast to the approach of Mohais et al., Clerc recommends to use a fixed *in-degree* for the neighborhood graph. This way, a particle is expected to have only a few neighbors, which is beneficial for problems with many local optima. There is, however, a non-zero probability of large neighborhood sizes. Experimental results on selected test functions suggest to set the in-degree to $K = 3$ [Cle06b].

2.2.3 Stereotyping

In 2000, Kennedy proposed a new approach for computing a particle's private and local guide, called *stereotyping* [Ken00]. The algorithm is inspired by the social-psychological assumption that individuals identify themselves with their social group and that they evaluate others by taking the respective social environment into account.

Moreover, social-psychologic studies indicate that opinions, beliefs, behavior, and norms of a social group tend to converge to the members' average opinions, beliefs and behavior [Ken00]. This convergence point needs not to be present as the attitudes of one of its members.

Based on these findings, Kennedy suggests to replace a particle's private and/or local guide by the respective group average. Therefore, all private guides are partitioned into several (e.g., 5) clusters based on their search space positions: The less the search space distance of two private guides is the more probable they are put into the same cluster. Any clustering algorithm can be used for this purpose, e.g., as proposed by Kennedy, *k-means clustering* [Llo82, KMN+02]. Three different approaches were investigated:

- When evaluating a particle's new velocity, the private guide's cluster center is used instead of the particle's private guide.
- When evaluating a particle's new velocity, the cluster center of its best neighbor is used instead of its local guide.
- For both the private and the local guide the respective cluster centers are used.

Based on his experimental study, Kennedy concluded that the first variant can remarkably improve a particle swarm's performance. However, especially the second method may lead to a performance decrease [Ken00].

2.2.4 Fully Informed Particle Swarm (FIPS)

Similar to Stereotyping presented in the previous section, in a *fully informed particle swarm (FIPS)* [MKN04, Men04, MKN03] each particle takes knowledge not only from its best, but from all its neighbors into account to compute its subsequent velocity and position. However, the approach is different: Instead of computing cluster centers, the information is directly incorporated into the velocity update equation.

Applying simple algebraic manipulations, Eq. (2.1) and Eq. (2.2) can be transformed to the following PSO update equations, which use the so-called *Type 1" constriction coefficient* χ of Clerc and Kennedy [CK02]:

$$\vec{v}_{i,t} = \chi \cdot \Big(\vec{v}_{i,t-1} + \varphi_1 \cdot \vec{r}_{1,i,t} \odot (\vec{p}_{i,t-1} - \vec{x}_{i,t-1}) + \varphi_2 \cdot \vec{r}_{2.i,t} \odot (\vec{l}_{i,t-1} - \vec{x}_{i,t-1})\Big) \quad (2.4)$$

$$\vec{x}_{i,t} = \vec{x}_{i,t-1} + \vec{v}_{i,t} \quad (2.5)$$

with $\chi = \omega$ and $\chi\varphi_i = c_i \Leftrightarrow \varphi_i = c_i/\chi$ for $i = 1,2$. Clerc and Kennedy proved, under simplifying assumptions, that a particle swarm converges, if χ and $\varphi = \varphi_1 + \varphi_2$ satisfy the following relation:

$$\chi = \frac{2 \cdot \kappa}{\varphi - 2 + \sqrt{\varphi^2 - 4 \cdot \varphi}} \quad (2.6)$$

where $\kappa \in [0,1]$ and $\varphi > 4$. Often, $\kappa = 1$, $\varphi_1 = \varphi_2 = 2.05$, i.e., $\varphi = 4.1$ and $\chi = 0.72984$, are used as a standard setting [BK07]. Interestingly, the convergence property is not dependent on the values of φ_1 and φ_2 but only on the sum these two coefficients. Hence, any number k of terms may contribute to a particle's velocity update without losing the convergent behavior, as long the coefficients $\varphi_1, \varphi_2, \dots \varphi_k$ sum up to an appropriate value [MKN04].

Mendes et al. [MKN04, Men04] propose three methodologies to include the information gathered from all adjacent individuals to a particle's movement: an unweighted approach, a fitness based approach and a distance based approach. For the unweighted approach, the velocity update is modified as follows:

$$\vec{v}_{i,t} = \chi \left(\vec{v}_{i,t-1} + \sum_{j \in \mathcal{N}_{i,t-1}} \frac{\varphi}{|\mathcal{N}_{i,t-1}|} \cdot \vec{r} \odot (\vec{p}_{j,t-1} - \vec{x}_{i,t-1}) \right)$$

where $\vec{r}$ is a vector of random real numbers between 0 and 1, which are regenerated every time they occur, and $\mathcal{N}_{i,t}$ is the set of indices of particle i's neighbors at time step t.

In the fitness based approach, better neighbors have greater influence on a particle's movement. Mendes considered minimization problems with positive fitness values, and weighted each neighbor's influence indirectly proportional to its private guide's fitness value: $1/f(\vec{p}_{j,t-1})$, where j is the index of the respective neighbor [Men04]. However, the fitness based version did not deliver better solutions than the unweighted FIPS in an experimental investigation [MKN04]. Therefore, mostly the unweighted FIPS is used because it is more efficient and simpler. The distance based approach provided worse results than the other two approaches and a standard PSO [MKN04, Men04], and is therefore seldomly regarded in the literature.

When introducing the fully informed particle swarm, Mendes et al. [MKN03, MKN04] reported that the algorithm provides very good results compared to a standard particle swarm optimizer. However, their experimental study showed that the neighborhood topology strongly influences its performance. Often, the less neighbors a particle has, the better are the results delivered by a fully informed particle swarm. The weighted FIPS in combination with a ring topology and self excluded succeeded to find solutions of predefined problem-dependent quality very close to the global optima in 100% of the runs on a testbed of six commonly-used benchmark functions, 200,000 function evaluations per run, and 40 runs per setting. However, the solution quality after 1,000 iterations (20,000 function evaluations) was still quite poor. Considering the achieved performance in a shorter time frame, the unweighted FIPS with grid topology and self excluded yielded the best results. Additionally, this setting was able to reach the predefined problem-dependent limit 98,9% of the cases [MKN03, MKN04]. It has to be noted that using densely connected neighborhood graphs, e.g., a fully connected one, can considerably deteriorate the performance of the FIPS algorithm [MKN03, MKN04].

Further studies with randomly generated graphs confirmed these observations. Mendes and Neves [MN04] investigated the effects of 3,289 neighborhood graphs with different characteristics on the performance of a fully informed particle swarm. They concluded that neighborhood graphs with an average degree of 4 and a low *clustering coefficient*[2] are best suited for the use with the FIPS algorithm.

In a later study, Kennedy and Mendes [KM06] tested the impact of 1,343 randomly generated neighborhood graphs with different characteristics on the achieved solution quality of both the FIPS algorithm and a standard particle swarm optimizer. They investigated 231 graphs with an average degree of $k = 3$, 311 graphs with an average degree of $k = 5$, and 801 graphs with an average degree of $k = 10$. The presented experimental results show that the FIPS algorithm performs best in combination with a neighborhood topology of low degree ($k = 3$), and that performance can be seriously deteriorated when using a denselier connected graph ($k = 10$).

Summarized, the FIPS algorithm with ring or grid topology often provided very good results, especially if a particle is not included in its own neighborhood. More densly connected topologies should not be used for a fully informed particle swarm.

2.2.5 Ranked FIPS

As described above, Mendes et al. [Men04, MKN04] proposed three different versions for the fully connected particle swarm. In the fitness based approach, the influence of each neighbor j is weighted with $1/f(\vec{p}_{j,t-1})$ [Men04], whereas in the unweighted approach, each adjacent particle has equal weight. Intuitively, the fitness based approach should yield better results as more problem-specific information is used for the particles' movement. However, in experimental investigations carried out by Mendes et al. [MKN03, MKN04], both approaches showed similar performance.

The reason might be that particles which are close to each other in the neighborhood graph often have similar fitness values as they are searching in the same search space regions. Hence, the weighting effect is diminished in comparison to the randomization [JHW08].

In order to overcome this problem, the *Ranked FIPS* was developed [JHW08]. In this variant, each particle ranks its neighbors according to the fitness values of their private guides. The lower the fitness value, the lower is a particle's rank. For the velocity update, the relative influence of each neighbor is determined according to its rank: The weight of an adjacent particle with rank r is twice as high as the weight of the neighbor with the subsequent rank $r+1$. The rank weights sum up to 1. Hence, as $\sum_{i=1}^{\infty} 1/2^i = 1$, if a particle had infinity many neighbors, the best particle would still be weighted with factor $1/2$. The less neighbors a particle has, the higher weights

[2]The clustering coefficient measures the average percentage of a particle's neighbors that are neighbors to one another.

are used, yielding the rank weights $r_1 \in [\frac{1}{2}, \frac{2}{3}]$, $r_2 \in [\frac{1}{4}, \frac{1}{3}]$, and so on, assuming that a particle has at least two neighbors. Hence, strong emphasis is given to the best neighbor even if a densely connected topology like the fully connected swarm is used.

Let again $\mathcal{N}_{i,t}$ be the set of indices of particle i's neighbors at time step t, then the ranked FIPS without private guide uses the following velocity update equation for particle i (adapted from [JHW08]):

$$\vec{v}_{i,t} = \chi \left(\vec{v}_{i,t-1} + \sum_{j=1}^{|\mathcal{N}_{i,t-1}|} r_j \cdot \varphi \cdot \vec{r} \odot (\vec{p}_{invrank_{i,t-1}(j),t-1} - \vec{x}_{i,t-1}) \right)$$

where the ranking function $invrank_{i,t-1} : [1, \ldots, |\mathcal{N}_{i,t-1}|] \rightarrow \mathcal{N}_{i,t-1}$ defines the (inverse) ranking on particle i's neighbors, i.e., each rank is mapped to the index of the respective neighbor according to the fitness values of their private guides as explained above. The rank weights $r_1, \ldots r_{|\mathcal{N}_{i,t-1}|}$ are given by

$$\begin{array}{rcl} \forall\ j = 1, \ldots, |\mathcal{N}_{i,t-1}| - 1 : \quad r_j & = & 2 \cdot r_{j+1} \\ \sum_{j=1}^{|\mathcal{N}_{i,t-1}|} r_j & = & 1 \end{array}$$

The ranked FIPS was compared to the unweighted FIPS and the fitness based FIPS on both a series of traditionally used benchmarks and on the recently introduced CEC 2005 benchmark suite. On most functions, the ranked FIPS yielded superior results than the other two variants [JHW08], and can therefore be considered as valuable alternative to other FIPS approaches.

2.3 PSO for Constrained Problems

In constrained optimization, solutions must satisfy a number of so-called *constraints*, which either restrict the parameter values to certain intervals or define dependencies among them. Formally, the task of constrained optimization in continuous domain is defined as follows:

$$\begin{array}{lllll} \text{Minimize} & f(\vec{x}) & & \\ \text{Subject to} & g_i(\vec{x}) \leq 0 & i = 1, \ldots, m_1 & \textit{(inequality constraints)} \\ & h_j(\vec{x}) = 0 & j = 1, \ldots, m_2 & \textit{(equality constraints)} \\ & lb_k \leq x_k \leq ub_k & k = 1, \ldots, n & \textit{(box constraints)} \end{array} \qquad (2.7)$$

The objective function f maps the n-dimensional parameter space $\mathcal{S} = [lb_1, ub_1] \times [lb_2, ub_2] \times \cdots \times [lb_n, ub_n] \subseteq \mathbb{R}^n$ to $\mathbb{R}$. The *feasible region* $\mathcal{F} \subset \mathbb{R}^n$ is given by the intersection of $\mathcal{S}$ with the equality and inequality constraints. The goal is to find a global optimal solution $\vec{x}^* \in \mathcal{F}$ with $\forall\, \vec{x} \in \mathcal{F} : f(\vec{x}^*) \leq f(\vec{x})$.

There exist a lot of strategies to cope with constraints in evolutionary algorithms and particle swarm optimization [MS96, Coe02b], which are outlined in the following.

2.3.1 Constraint Handling

The available constraint handling techniques can be classified into the following five categories [Coe02b]:

- *Penalty functions*
- *Repair algorithms*
- *Special representations and operators (including decoders)*
- *Separation of objectives and constraints*
- *Hybrid methods*

Penalty Functions

The use of penalty functions is one of the most common approaches to deal with constraints in evolutionary computation. If a minimization problem is assumed, the objective function f is modified to f_{new} [Coe02b]:

$$f_{new}(\vec{x}) = f(\vec{x}) + \sum_{i=1}^{m_1} a_i \cdot \max\{0, g_i(\vec{x})\}^{\alpha} + \sum_{j=1}^{m_2} b_j \cdot |h_j(\vec{x})|^{\beta}$$

where α, β, and the *penalty coefficients* $a_1, \dots, a_{m_1}, b_1, \dots, b_{m_2} > 0$ are user-defined parameters, which have to be selected carefully. If the penalty coefficients are chosen too low, the population might explore infeasible space most of the time while when chosen too high, the individuals are distracted from the boundary and might be unable to locate disconnected feasible regions [Coe02b]. Besides being statically defined at the beginning of the optimization and used throughout the run, the penalty coefficients can be dynamically adapted during the process. Among the dynamic approaches, there are time-dependent ones, which means that the penalty coefficients increase with the number of iterations to guide the population to feasible regions at later stages of the optimization. Other dynamic approaches adapt the penalty coefficients to the optimization process by taking the number of feasible and infeasible individuals into account, by trying to stress the importance of difficult constraints, or by using co-evolutionary approaches [MS96, Coe02b].

A conceptually different approach is the use of so-called *death penalties* (see, e.g., [MS96]). This means that infeasible individuals are either completely rejected or regenerated until they satisfy all constraints. Note that the latter alternative might be a very time-consuming task in the occurence of difficult constraints.

Penalty function approaches can be utilized in combination with PSO algorithms as well when solving constrained optimization problems. Parsopoulos and Vrahatis [PV02b] used a dynamic, time-dependent penalty function for particle swarm optimization, and reported promising results on six well-known constrained benchmark problems. Static penalty functions were successfully used for the optimization of electrical power systems with PSO algorithms [Abi02, YKF+00].

Repair Algorithms

Another approach to cope with infeasible solutions in constrained optimization is to map each $\vec{x} \in \mathcal{S}$ to a feasible solution $\vec{z} \in \mathcal{F}$. This process is also denoted as *repairing* the infeasible solution $\vec{x}$. More formally, a *repair function* $f_{repair} : \mathcal{S} \rightarrow \mathcal{F}$ is defined, and the objective function is modified as follows:

$$f_{new}(\vec{x}) = f(f_{repair}(\vec{x})) \quad .$$

There are two aspects which have to be considered when designing repair functions: First, f_{repair} can be either deterministic or stochastic. When randomness is involved, two evaluations of f_{repair} with the same search space element $\vec{x}$ might produce two different results $\vec{z}_1, \vec{z}_2 \in \mathcal{F}, \vec{z}_1 \neq \vec{z}_2$. Second, the repaired solution can either be used only for the fitness evaluation of an individual, i.e., that the objective function is modified as stated above, or an infeasible individual might be replaced (with some probability) by its repaired version [Mic00c, Coe02b].

In some cases, it is relatively simple to repair infeasible solutions. When solving, for example, the knapsack problem, items may be removed from the knapsack, either in a deterministic or in a stochastic way, until the solution is feasible [Mic00c]. However, in general, the design of an efficient repair function is a non-trivial task.

In particle swarm optimization, repair methods are often utilized to satisfy box constraints. Usually, particles are replaced with their repaired version. A variety of deterministic and stochastic repair algorithms to handle box constraints exist in the literature (see, for instance, [Cle06a, ABEF05]). Due to their importance for this work, they are presented in detail in Section 2.3.2.

Repair strategies can be used in combination with other constraint handling mechanisms as well. E.g., in Genocop III, an evolutionary optimization system presented by Michalewicz and Nazhiyath [MN95], specialized mutation and crossover operators guarantee that all linear constraints are satisfied during the optimization. However, non-linear constraints are handled by either a stochastic or a deterministic repair procedure [Mic00c]. In the so-called *quantum particle swarm optimizer*, box constraints are satisfied using a stochastic repair mechanism while for all other constraints a static penalty function is applied [dS08].

Special Representations and Operators

In this category, constrained optimization problems are tackled by either

- the use of specialized operators which keep the population inside the feasible space (e.g., modified position and velocity update equations), or

- the modification of the problem's representation such that the constraints can be satisfied more easily (e.g., by simplifying the shape of the search space).

In Genocop III, specialized mutation and crossover operators guarantee the feasibility of the generated solutions when solving linearly constrained problems [MN95]. Let $\vec{x}_1$ and $\vec{x}_2$ be two feasible solutions, then arithmetic crossover $\vec{y} = a \cdot \vec{x}_1 + (1-a) \cdot \vec{x}_2, a \in [0,1]$, results in a feasible offspring $\vec{y}$. A modified mutation operator is applied as well.

Similarly, in particle swarm optimization linear equality constraints can be satisfied by using random values instead of random vectors in the velocity update equation [PE03]:

$$\begin{aligned} \vec{v}_{i,t} &= \omega \cdot \vec{v}_{i,t-1} + c_1 \cdot r_{1,i,t} \cdot (\vec{p}_{i,t-1} - \vec{x}_{i,t-1}) + c_2 \cdot r_{2,i,t} \cdot (\vec{l}_{i,t-1} - \vec{x}_{i,t-1}) \\ \vec{x}_{i,t} &= \vec{x}_{i,t-1} + \vec{v}_{i,t} \end{aligned}$$

In this case, the new particle position is a linear combination of feasible solutions, and therefore also feasible. In order to avoid premature convergence and to ensure the reachability of all feasible solutions, a specialized mutation operator was implemented by Paquet and Engelbrecht [PE03]. Note, however, that linear inequality constraints are not necessarly fulfilled. Therefore, Halter and Mostaghim extended this approach with a repair mechanism to deal with general linearly constrained optimization problems [HM06].

Instead of designing specialized operators which preserve the feasibility of the generated solutions, the problem's representation can be modified such that it is easier to handle the constraints. An example of such a transformation are *decoders* [Mic00a]. Formally, decoders can be described by a decoder function $f_{dec} : \mathcal{R} \rightarrow \mathcal{F}$ which maps a representation space $\mathcal{R}$ to the feasible solutions. The representation space can be an arbitrary space which is explored by the population of a stochastic search algorithm, but should, of course, have a simpler shape than the original feasible space.

When utilizing decoders, it is important that each feasible solution $\vec{x} \in \mathcal{F}$ can be reached, i.e., that f_{dec} is surjective, that each $\vec{x} \in \mathcal{F}$ has the same number of representations, that the decoder function can be computed efficiently, and that small changes in the representation space result in small changes in the feasible space [Mic00a].

Koziel and Michalewicz presented a decoder-based approach with $\mathcal{R} = [-1,1]^n$ which can be used for any constrained optimization problem in continuous domain. For convex spaces, the decoder function $f_{dec} : [-1,1]^n \rightarrow \mathcal{F}$ is defined as [KM98]:

$$f_{dec}(\vec{y}) = \begin{cases} \vec{r}_0 + y_{max} \cdot t_0 \cdot \underbrace{\left(f_{\mathcal{S}}\left(\frac{\vec{y}}{y_{max}} \right) - \vec{r}_0 \right)}_{\vec{l}} & \text{if } \vec{y} \neq \vec{0} \\ \vec{r}_0 & \text{otherwise} \end{cases} \tag{2.8}$$

where the $f_{\mathcal{S}} : [-1,1]^n \rightarrow \mathcal{S}$ intuitively maps the n-dimensional cube $[-1,1]^n$ to the rectangularly bounded n-dimensional search space, $\vec{r}_0 \in \mathcal{F}$ is an arbitrary feasible reference point, $y_{max} = \max_{i=1\ldots n} |y_i|$, and t_0 is computed by binary search such that $r_0 + t_0 \cdot \vec{l}$ is located on the boundary of $\mathcal{F}$. Note that t_0 is unique for convex feasible spaces. The homomorphous mapping is illustrated in Figure 2.5. Koziel and Michalewicz extended their approach for non-convex spaces [KM98, KM99]. However, an additional parameter is needed in that case.

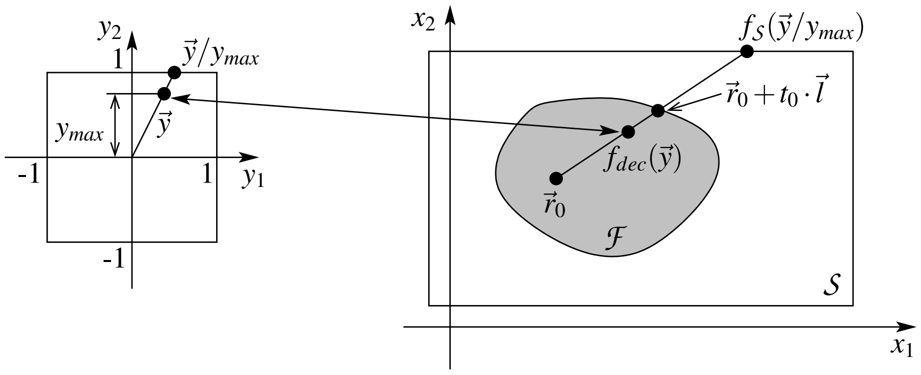


Figure 2.5: Illustration of the homomorphous mapping as defined by Koziel and Michalewicz [KM98, KM99]. The decoder function $f_{dec} : [-1,1]^n \rightarrow \mathcal{F}$ and $\vec{l}$ are given in Equation (2.8).

Monson and Seppi [MS05] used a homomorphous mapping for optimization problems that solely incorporate linear equality constraints. Let $\mathcal{S} = \mathbb{R}^n$ be the n-dimensional search space of the optimization problem. Then, the equality constraints define a lower-dimensional space, which is transformed such that the explored representation space is $\mathbb{R}^m$ with $m \leq n$.

Some particle swarm optimizers cope with constrained optimization problems by transforming them in unconstrained ones. Hu, Eberhart and Shi [HE02b, HES03a] allow particles to leave $\mathcal{F}$, but they neither evaluate infeasible solutions nor consider them in the private guide update. For minimization problems, this method is equivalent to modifying the objective function to:

$$f_{new}(\vec{x}) = \begin{cases} f(\vec{x}) & \text{if } \vec{x} \in \mathcal{F} \\ +\infty & \text{otherwise} \end{cases}$$

In the PSO variant of Zhang et al. [ZXB04], box constraints are eliminated by constructing infinitely many copies of the n-dimensional search space which are placed side by side in $\mathbb{R}^n$ as illustrated in Figure 2.6.

Separation of Objectives and Constraints

Some constraint handling techniques aim at optimizing constraints and objective function seperately. There exist several approaches in this category, for instance:

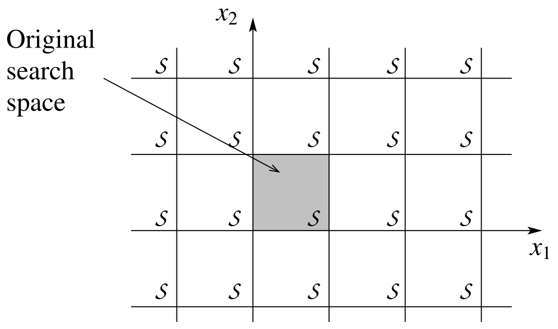


Figure 2.6: Periodic search space of Zhang et al. [ZXB04] for a two-dimensional problem.

- The problem can be transformed into a multi-objective optimization problem (see, e.g., [Mic00b] for details).
- Deb [Deb00] used a modified selection scheme for handling constraints in evolutionary computation. Whenever two solutions compete, a feasible solution is preferred to an infeasible one. If both solutions are feasible, they are compared with respect to their objective values, whereas if both are infeasible, the summarized constraint violations are considered. A similar approach was presented by Pulido and Coello for particle swarm optimization [PC04].
- Takahama and Sakai [TS06] proposed a so-called ε-constrained PSO with a modified order relation for the solutions produced by a particle swarm. For the comparison of two candidate solutions, the objective value is only used if either the constraint violations are equal (note that a definition for the computation of constraint violations is needed), or if the constraint violations of both solutions are smaller than a specified, possibly dynamic, threshold ε. Otherwise, the constraint violations are used for the comparison.
- Liang and Suganthan [LS06] extended their dynamic multi-swarm optimizer [LS05] for constrained optimization problems by adaptively assigning subswarms to solve different constraints, and by utilizing specialized criteria for the comparison of two particles.

2.3.2 Bound Handling

Many optimization problems have at least box constraints, i.e., the search space $\mathcal{S} = [lb_1, ub_1] \times [lb_2, ub_2] \times \cdots \times [lb_n, ub_n]$ is bounded. Even if constraints are eliminated by certain approaches, e.g., the homomorphous mapping of Koziel and Michalewicz [KM98, KM99], box constraints are not removed. Feasible solutions can be located efficiently when solving box-constrained problems. In PSO algorithms, box constraints are usually tackled by repairing infeasible solutions, or by using a modified problem representation or velocity operator. Note that repair algorithms have to consider both position and velocity handling. Moreover, general constraint handling techniques like penalizing infeasible solutions as discussed in the previous section can be applied.

Repair Algorithms – Position Handling

The application of the standard PSO update equations does not prevent particles from leaving the specified search space. In this section, repair mechanisms, which map infeasible particle positions to feasible ones, are discussed. When repairing a particle, it often makes sense to also modifiy its velocity. Assume, for instance, that infeasible particles are set to the nearest boundary:

$$x_{i,t,d} = \begin{cases} lb_d & \text{if } x_{i,t,d} < lb_d \\ ub_d & \text{if } x_{i,t,d} > ub_d \\ x_{i,t,d} & \text{otherwise} \end{cases}$$

where $x_{i,t,d}$ is the d-th component of particle i's position at time step t. If $x_{i,t,d} > ub_d$ and the velocity is not altered, $v_{i,t,d} \geq 0$ holds and the particle tends to leave the search space again (similarly, for $x_{i,t,d} < lb_d$). Hence, it would make sense to, for example, set the velocity to zero or even invert the d-th velocity component so that the particle is pulled back into the search space. Velocity handling strategies are discussed in the next paragraph. For position handling several repair strategies exist (see also Figure 2.7):

- *Nearest:* Infeasible particles are reset to the nearest boundary as defined above [CPL04, BM06, ZXB04, Cle06a, Cle06b].
- *Shrink:* The particle position is set to the intersection point of $\vec{x}_{i,t} - \vec{x}_{i,t-1}$ with the boundary (i.e., the particle "stops" at the boundary) [ABEF05, Eng05]. Hence, the particle position is computed as

  $$\vec{x}_{i,t} = \vec{x}_{i,t-1} + \sigma_{i,t} \cdot \vec{v}_{i,t}$$

  with

  $$\sigma_{i,t,d} = \begin{cases} (lb_d - x_{i,t-1,d})/v_{i,t,d} & \text{if } x_{i,t,d} < lb_d \\ (ub_d - x_{i,t-1,d})/v_{i,t,d} & \text{if } x_{i,t,d} > ub_d \\ 1 & \text{otherwise} \end{cases}$$

  and

  $$\sigma_{i,t} = \min_{d=1\ldots n} \{\sigma_{i,t,d}\} \ .$$

  This method can also be utilized for general linear constraints [HM06].

- *Random:* Each $x_{i,t,d} \notin [lb_d, ub_d]$ is redrawn uniformly at random between lb_d and ub_d [ZXB04, Eng05].

- *Reflect:* Infeasible particles are reflected at the boundary [BF05]. This process is repeated until the particle position satisfies the box constraints:

```
while $x_{i,t,d} \notin [lb_d, ub_d]$ do
   if $x_{i,t,d} < lb_d$ then
      $x_{i,t,d} = 2 \cdot lb_d - x_{i,t,d}$
   else if $x_{i,t,d} > ub_d$ then
      $x_{i,t,d} = 2 \cdot ub_d - x_{i,t,d}$
   end if
end while
```

- *Intermediate:* If $x_{i,t,d} > ub_d$ then $x_{i,t,d}$ is set to an intermediate value of $x_{i,t-1,d}$ and ub_d (the case $x_{i,t,d} < lb_d$ is handled analogically). Alvarez-Benitez et al. [ABEF05] used a truncated exponential distribution that prefers boundary regions for this purpose. The utilization of other probability distributions is also possible. Zielinski et al. [ZWLK09] deterministically set infeasible particles to the center of $x_{i,t-1,d}$ and ub_d (resp. lb_d), i.e., $x_{i,t,d} = \frac{1}{2}(x_{i,t-1,d} + ub_d)$ if $x_{i,t,d} > ub_d$.

- *Resample:* The stochastic components in the velocity update equation are redrawn until the particle position is feasible [ABEF05].

Repair Algorithms – Velocity Handling

Whenever an infeasible particle's position is repaired, it might make sense to also modify its velocity. Typical velocity handling strategies are:

- *Unmodified:* Both feasible and infeasible particles keep their velocity.

- *Adjust:* After position handling, the velocity is adjusted such that $\vec{v}_{i,t} = \vec{x}_{i,t} - \vec{x}_{i,t-1}$ holds.

- *Zero:* Let $\mathcal{D}_{i,t} = \{d \mid x_{i,t,d} \notin [lb_d, ub_d]\}$ be a set of indices, and $\vec{v}_{i,t}$ the velocity vector of particle i at time step t. Then $\forall d \in \mathcal{D}_{i,t} : v_{i,t,d} = 0$. In other words, $v_{i,t,d}$ is set to zero iff particle i crosses the boundary in dimension d at time step t. This method can be utilized with or without a position handling strategy [Cle06b, Cle06a, Eng05, RRS04].

- *Invert:* Let $\mathcal{D}_{i,t}$ be defined as above, then $\forall d \in \mathcal{D}_{i,t} : v_{i,t,d} = -z \cdot v_{i,t,d}$, where $z \geq 0$ is a user-defined parameter. This method is usually applied in combination

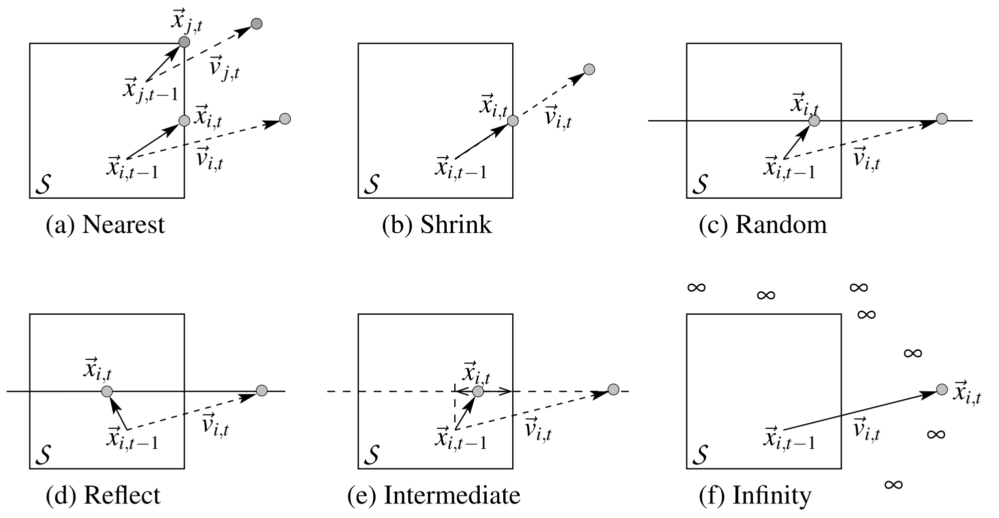


Figure 2.7: Various position handling strategies utilized in particle swarm optimization, illustrated for two-dimensional search spaces $\mathcal{S}$. Figures (a)–(e) show repair strategies whereas Figure (f) depicts a special representation of the problem: Infeasible particle positions evaluate to $+\infty$ for minimization problems.

with a bound resetting method such as *Nearest* in order to pull particles back into the search space. Typical values for z are $z = 0$, which is equivalent to the *Zero* strategy, $z = 0.5$ [Cle06a], and $z = 1$ [CPL04, MK05]. Alternatively, z may be drawn at random according to a specified probability distribution, independently for each component [Cle06a].

Special Representations

Instead of utilizing repair strategies, infeasible particle positions can be accepted if the objective function is modified appropriately. Often, the objective function value is set to infinity for all $\vec{x} \notin \mathcal{S}$ [Men04, BK07, Ken07, Eng05, RRS04]. This method is equivalent to not evaluate infeasible particles and equivalent to prevent that a particle's private guide is updated to an infeasible solution. Using the previously presented categories for constraint handling techniques, this approach can be seen as a special representation of the problem. For minimization problems, the objective function is modified to:

$$f_{new}(\vec{x}) = \begin{cases} f(\vec{x}) & \text{if } \vec{x} \in \mathcal{S} \\ +\infty & \text{otherwise} \end{cases}$$

This strategy, which will be denoted as *Infinity* henceforth, is recommended by Kennedy, one of the inventors of the PSO algorithm, due to the following reasons [BK07, Ken07]: It is straightforward, easy to apply, and the trajectories of the particles are not disturbed. Kennedy argues that there is no reason to limit the particle positions to a specified range as the skipped evaluation can be used in a later iteration.

Zhang et al. [ZXB04] also accept infeasible particle positions. In their approch the n-dimensional search space is copied infinite many times and placed side by side to cover $\mathbb{R}^n$ (see Figure 2.6).

Special Velocity Update Operators

The goal of special velocity update operators is either to to preserve the feasibility of the particles or to help them to locate feasible regions.

The so-called *Hyperbolic* method presented by Clerc [Cle06a] is a feasibility preserving approach. The d-th component of particle i's position and velocity vector is updated using the following equations:

$$v_{i,t,d} = \omega \cdot v_{i,t-1,d} + c_1 \cdot r_{1,i,t,d} \cdot (p_{i,t-1,d} - x_{i,t-1,d}) + c_2 \cdot r_{2,i,t,d} \cdot (l_{i,t-1,d} - x_{i,t-1,d})$$

```
if $v_{i,t,d} > 0$ then
    $v_{i,t,d} = \frac{v_{i,t,d}}{1 + \frac{v_{i,t,d}}{ub_d - x_{i,t-1,d}}}$
else
    $v_{i,t,d} = \frac{v_{i,t,d}}{1 - \frac{v_{i,t,d}}{x_{i,t-1,d} - lb_d}}$
end if
```

$$x_{i,t,d} = x_{i,t-1,d} + v_{i,t,d} \tag{2.9}$$

After applying the standard velocity update, the velocity is adjusted such that the resulting particle position is feasible, as shown in Figure 2.8 for a one-dimensional search space $\mathcal{S} = [-100, 100]$.

The definition of a maximum velocity for each component, also denoted as *velocity clamping*, is another special velocity operator which might help particles to reenter the search space once they are infeasible. Kennedy suggests to rather use velocity clamping than position clamping in the occurence of box constraints [Ken07]. Let $\mathcal{V} = [-V_{max,1}, V_{max,1}] \times [-V_{max,2}, V_{max,2}] \times \cdots \times [-V_{max,n}, V_{max,n}]$ then the standard velocity update equation is followed by:

```
if $v_{i,t,d} > V_{max,d}$ then
    $v_{i,t,d} = V_{max,d}$
else if $v_{i,t,d} < -V_{max,d}$ then
    $v_{i,t,d} = -V_{max,d}$
end if
```

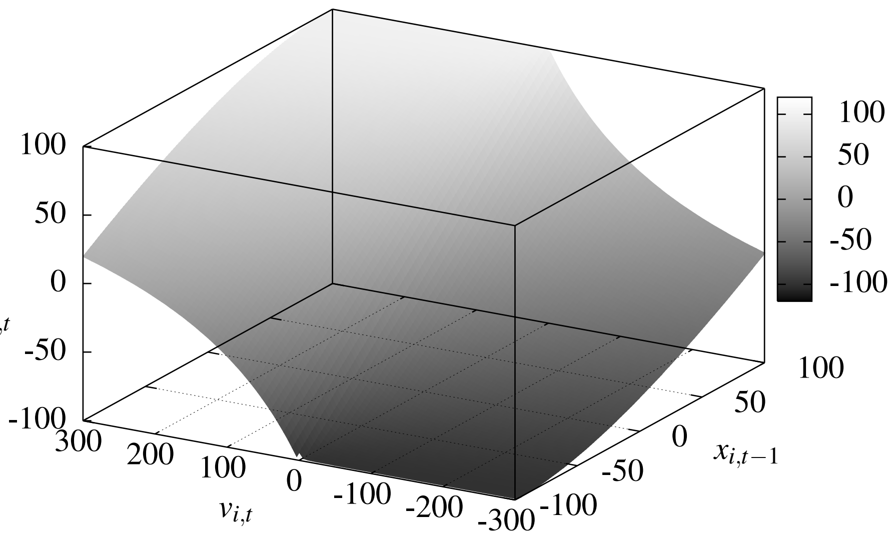


Figure 2.8: *Hyperbolic* boundary handling as presented by Clerc [Cle06a]. Particle i's new position $x_{i,t}$ is shown in dependence of the old position $x_{i,t-1}$ and the updated (but not yet corrected) velocity $v_{i,t}$. Obviously, $x_{i,t} \in [-100, 100]$, which would also be true for broader ranges of $v_{i,t}$.

2.4 Theoretical Results for Particle Swarm Optimization

Although the core of the PSO algorithm can be described by two formulas (velocity and position update equation), the exact swarm behavior and characteristics such as convergence properties or sampling distributions are by far not obvious. In fact, the analysis of particle swarm optimization is a great challenge due to the particles' interactions and the stochastic nature of the velocity update, which make it difficult to find accurate models reflecting the behavior of a real particle swarm. However, progress was made in recent years, leading to a better understanding of the swarm's dynamics and providing parameter selection guidelines based thereon. Before presenting the models that are utilized in the literature of PSO analyses, the standard position and velocity update equations (cf. Eq. (2.1) and Eq. (2.2), page 9) are repeated here for convenience:

$$\begin{aligned}\vec{v}_{i,t} &= \omega \cdot \vec{v}_{i,t-1} + c_1 \cdot \vec{r}_{1,i,t} \odot (\vec{p}_{i,t-1} - \vec{x}_{i,t-1}) + c_2 \cdot \vec{r}_{2,i,t} \odot (\vec{l}_{i,t-1} - \vec{x}_{i,t-1})\\ \vec{x}_{i,t} &= \vec{x}_{i,t-1} + \vec{v}_{i,t}\end{aligned}$$

2.4.1 Models

Deterministic model [vdB02, CK02, Tre03, vdBE06, OM99, YII03]. As each component of a particle's position and velocity vector is updated independently, a one-dimensional problem is considered without loss of generality (see, e.g., [vdB02]). Furthermore, stagnation is assumed, which means that none of the particles improves its personal best. This implies that all particles move independently from one another through the search space, and the subscript i can be dropped [JLY07a, PB07][3]. Due to the stagnation, a particle's private and local guide are constants, and will be denoted as p and l in the following. By removing the stochastic components, the following one-dimensional deterministic PSO update equations are derived:

$$v_t = \omega \cdot v_{t-1} + \phi_1 \cdot (p - x_{t-1}) + \phi_2 \cdot (l - x_{t-1}) \tag{2.10}$$

$$x_t = x_{t-1} + v_t \tag{2.11}$$

where ϕ_1 and ϕ_2 are constants. By algebraic manipulations[4], the system can be transformed to a non-homogeneous linear recurrence relation [vdB02, vdBE06, OM99]:

$$x_t = (1 + \omega - \phi_1 - \phi_2) \cdot x_{t-1} - \omega \cdot x_{t-2} + \phi_1 p + \phi_2 l \tag{2.12}$$

As an alternative notation, the simplified PSO algorithm can be written as a linear, discrete-time dynamic system [CK02, Tre03, YII03] with states $\vec{y}_t$ and system matrix A such that $\vec{y}_t = A \cdot \vec{y}_{t-1} + \vec{z}$. There are different possibilities to define $\vec{y}_t$, A, and $\vec{z}$, for instance (similar to [Tre03]):

$$\vec{y}_t = \begin{bmatrix} x_t \\ v_t \end{bmatrix}, \quad A = \begin{bmatrix} 1 - \phi_1 - \phi_2 & \omega \\ -\phi_1 - \phi_2 & \omega \end{bmatrix}, \quad \vec{z} = \begin{bmatrix} \phi_1 p + \phi_2 l \\ \phi_1 p + \phi_2 l \end{bmatrix} \tag{2.13}$$

Stochastic model [JLY07a, JLY07b, PB07, Pol08]. Similar to the deterministic model, a one-dimensional problem and stagnation are assumed. However, the stochastic parts of the velocity update equation are kept. Again, the system can be rewritten as a non-homogeneous linear recurrence relation [JLY07a, JLY07b, PB07, Pol08]:

$$x_t = (1 + \omega - c_1 r_{1,t} - c_2 r_{2,t}) \cdot x_{t-1} - \omega \cdot x_{t-2} + c_1 r_{1,t} p + c_2 r_{2,t} l \tag{2.14}$$

In this relation, whilst c_1, c_2, ω, p, and l are constants, $r_{1,t}$, $r_{2,t}$, x_0, and x_1 are stochastic variables. The particle's trajectory $\{x_t\}$ is a stochastic process [JLY07a], from which expectation values and variances [JLY07a, JLY07b, PB07], or higher-order moments [Pol08] may be derived.

[3]Some authors consider particles in isolation [vdB02] or reduce the population to a single individual [CK02], which results in the same model due to the fact that they additionally assume that the isolated/single particle does not improve its private guide.

[4]Insert $v_t = x_t - x_{t-1}$ and $v_{t-1} = x_{t-1} - x_{t-2}$ which are obtained from Eq. (2.11) in Eq. (2.10).

Markov chain model [PLCS07, PL07]. The preceding models are typically analyzed without taking the optimization problem to be solved into account. Instead, stagnation is assumed, and therefore, the trajectories of the particles do not depend on the shape of the objective function. In 2007, Poli et al. [PLCS07] suggested to model stochastic search algorithms for continuous problems, such as evolution strategies or particle swarm optimization, by discrete Markov chains. Most evolutionary algorithms can be described such that the Markovian property, which means that the next state of the algorithm only depends on its current state, is fulfilled. Poli et al. discretisize the n-dimensional search space $\mathcal{S}$ into a finite number of compact non-overlapping subsets $\mathcal{S}_1, \ldots, \mathcal{S}_k$. To each sub-domain, a fitness value $f_i, i = 1, \ldots, k$ is assigned, e.g., $f_i = f(\vec{x}_{c_i})$, where $\vec{x}_{c_i}$ is the central point of $\mathcal{S}_i$. The state of an evolutionary algorithm is usually its population, i.e., a set of search space positions, and possibly additional algorithmic components such as private guides and velocities in a particle swarm optimizer. In order to obtain a finite number of states, an individual can, for instance, be represented by the index i of the respective sub-domain $\mathcal{S}_i$ it is located in.

Poli et al. [PLCS07, PL07] presented a Markov chain model for the so-called *bare bones particle swarm optimizer* [Ken03], which samples a particle's new position componentwise from a Gaussian distribution with mean $(p_{i,t-1,d} + l_{i,t-1,d})/2$ and standard deviation $|p_{i,t-1,d} - l_{i,t-1,d}|$. The state of this particular particle swarm optimizer can be described by the particles' private guides, as no additional information (such as a particle's current position) is needed for the update equation. Hence, the state of a bare bones PSO with five particles can, for instance, be expressed by $Y_t = (1,1,2,1,6)$, which means that three particles are located in sub-domain $\mathcal{S}_1$, and one particle is located in sub-domains $\mathcal{S}_2$ and $\mathcal{S}_6$, respectively, at time step t. When sub-domains are ordered such that a higher index reflects a better fitness value, the fifth particle (in sub-domain $\mathcal{S}_6$) currently carries the best private guide, and serves, using a fully connected neighborhood topology, as local guide for all other particles. With these definitions in mind, Poli et al. determined the state transition matrix M of different bare bones particle swarm optimizers [PLCS07, PL07]. E.g., the probability of transition $(1,1,2,1,6) \rightarrow (1,1,2,1,5)$ is zero as private guides never deteriorate during the optimization (and the fitness of $\mathcal{S}_5$ is worse than the fitness of $\mathcal{S}_6$ due to the order of the sub-domains). However, other state transitions are possible, e.g., switching from state $(1,1,2,1,6)$ to state $(1,1,2,5,6)$. The respective probabilities were determined by Poli et al. [PLCS07, PL07].

The Markov chain model is very powerful, as the probability that an algorithm is in a particular state at time step t can be determined by iterating the state transition matrix M. More formally, let N be the number of states and $\pi_{j,t}$ denote the probability that the algorithm is in state j at time step t. Then, $\vec{\pi}_t = (\pi_{1,t}, \pi_{2,t}, \ldots, \pi_{N,t})$ can be computed by $\vec{\pi}_t = \vec{\pi}_0 \cdot M^t$. However, the computation of $\vec{\pi}_t$ can be a very time-consuming task as M grows quadratically with the number of sub-domains and exponentially with the number of particles and dimensions [PL07].

2.4.2 Analyses and Results

Many PSO analyses are based on a deterministic or stochastic model in which the algorithm is either represented by a linear non-homogeneous recurrence relation or a linear dynamic system. The application of dynamic systems theory allows to study the convergence properties of the PSO model by computing the eigenvalues of the system matrix A. If and only if the magnitude of all eigenvalues is smaller than 1, the system converges to a stable point (see, e.g., [Tre03, YII03]). From this condition, parameter guidelines can be derived.

Analyses of Deterministic PSO Models

Ozcan and Mohan [OM99] analyzed the recurrence relation given in Eq. (2.12) with $\omega = 1$, and suggest to choose $c_1 + c_2 < 4$. Furthermore, they showed that the particles' trajectories in this simplified system can be described by functions of sine and cosine.

Van den Bergh et al. [vdB02, vdBE06] studied the recurrence relation given in Eq. (2.12) with respect to the convergence properties of the sequence $\{x_t\}_{t=0}^{+\infty}$. They formally defined convergence as $\lim_{t\to+\infty} x_t = x$, where x is the point of convergence. However, convergence of the series does not imply that x is a local or global optimum. By analyzing the deterministic model, Bergh et al. showed that a particle's trajectory is guaranteed to converge to the weighted average of its private and local guide if c_1, c_2, and ω are chosen such that the relation

$$\omega > \frac{1}{2}(c_1 + c_2) - 1 \quad \Leftrightarrow \quad c_1 + c_2 < 2(\omega + 1)$$

is fulfilled [vdB02, vdBE06]. A similar result was obtained by Yasuda et al. [YII03], however, they did not replace ϕ_1 and ϕ_2 with their upper bounds c_1 and c_2, yielding the following convergence criterion:

$$\phi_1 + \phi_2 < 2(\omega + 1)$$

In contrast to the previous approaches, ϕ_1 and ϕ_2 were replaced by their expectation values $c_1/2$ and $c_2/2$, respectively, in Trelea's PSO analysis. Hence, the following relationship between accelearation constants and inertia weight was established [Tre03]:

$$c_1 + c_2 < 4(\omega + 1) \tag{2.15}$$

Trelea not only provided convergence criteria, but additionally analyzed the kind of convergence, which depends on the characteristics of the eigenvalues of the system matrix, e.g., real-valued, complex, positive or negative. Conditions were derived to obtain oscillatory, either harmonic or zigzagging, or non-oscillatory convergence behavior.

Clerc and Kennedy [CK02] analyzed the PSO algorithm in both discrete and continuous time using a deterministic model similar to the one presented above. A so-called *constricted particle swarm optimizer* was derived by rewriting the original PSO equations to

$$\vec{v}_{i,t} = \chi \cdot \Big(\vec{v}_{i,t-1} + \varphi_1 \cdot \vec{r}_{1,i,t} \odot (\vec{p}_{i,t-1} - \vec{x}_{i,t-1}) + \varphi_2 \cdot \vec{r}_{2,i,t} \odot (\vec{l}_{i,t-1} - \vec{x}_{i,t-1})\Big) \tag{2.16}$$

$$\vec{x}_{i,t} = \vec{x}_{i,t-1} + \vec{v}_{i,t} \tag{2.17}$$

where χ is denoted as *constriction coefficient*. These update equations are algebraically equivalent to the standard ones (see Eq. (2.1) and Eq. (2.2)) by setting $\chi = \omega$ and $\chi\varphi_i = c_i \Leftrightarrow \varphi_i = c_i/\chi$ for $i = 1,2$. Clerc and Kennedy proved that the dynamic system converges if the *constriction coefficient* χ is computed as follows:

$$\chi = \frac{2 \cdot \kappa}{\varphi - 2 + \sqrt{\varphi^2 - 4 \cdot \varphi}} \tag{2.18}$$

where $\kappa \in [0,1]$ and $\varphi = \varphi_1 + \varphi_2 > 4$. The convergence speed can be controlled by κ: High values, e.g., $\kappa = 1$, correspond to slow convergence.

Analyses of Stochastic and Markov Chain PSO Models

Jiang, Luo and Yang [JLY07a, JLY07b] and Poli and Broomhead [PB07] considered the particles' trajectories as a stochastic process using the stochastic non-homogeneous recurrence relation given in Eq. (2.14). By applying the expectation operator on both sides of Eq. (2.14), an iterative process is obtained which can be analyzed by computing the roots of the associated characteristic polynom. The magnitudes of these roots must be smaller than one for a convergent system. The series of expectation values converges if the relation suggested by Trelea (Eq. (2.15)) is fulfilled [JLY07a]. However, convergence of the sequence of expectation values, $\{E[x_t]\}$, does not imply convergence of $\{x_t\}$. Convergence of $\{x_t\}$ is only guaranteed if additionally the sequence of standard deviations, denoted as $\{StdDev[x_t]\}$, approaches zero. This aspect was analyzed by Jiang et al., and conditions for a convergent sequence of variances (the sequence will be denoted as $\{Var[x_t]\}$) were derived [JLY07a]. The study was extended by not only defining convergence criteria, but by establishing relations between the eigenvalues of the iterative processes $\{E[x_t]\}$ and $\{Var[x_t]\}$ and the algorithmic parameters c_1, c_2, and ω. These relations allow to control the convergence speed of a particle swarm by appropriate parameter selection [JLY07b]. Poli and Broomhead [PB07] analyzed the sequence of standard deviations in the stochastic model. They showed that $\{StdDev[x_t]\}$ can only converge to zero if $p = l$. Hence, particles with $p \neq l$ keep on exploring. Note that a parameter setting that results in $\{StdDev[x_t]\} \not\to 0$ for all particles might be desirable to enhance a swarm's capability to escape local optima [PB07].

Poli [Pol08] studied the dynamics of higher-order moments of the stochastic sequence $\{x_t\}$ presented in Eq. (2.14), and showed that, theoretically, the respective recurrence relations can be obtained for moments of arbitrary order. Based on these findings, the density function of the sampling distribution of PSO during stagnation was approximated by using the first four moments. Experimentation showed that the model's predictions are extremely accurate [Pol08].

The so-called *bare bones particle swarm optimizer* [Ken03] was analyzed by Poli et al. [PLCS07, PL07] using the Markov chain model presented above. Success probability and expected runtime of different bare bones particle swarm optimizers was computed for several one-dimensional testfunctions. Again, the results are confirmed by experimental investigations [PLCS07, PL07].

Kadirkamanathan et al. [KSF06] analyzed the stability of a stochastic PSO model and derived sufficient conditions for asymptotic stability in the sense of Lyapunov stability.

Analyses of PSO Variants

The previously presented theoretical investigations either study a model of the standard particle swarm optimizer or the bare bones PSO. There exist some analyses for special PSO variants: Veeramachaneni et al. [VOK07] took first steps in analyzing the density function of a particle's velocity in the binary PSO of Kennedy and Eberhart. A theoretical runtime analysis of the same binary particle swarm optimizer was performed by Sudholt and Witt [SW08]. Poli et al. [PBBK07] analyzed the first four moments of a self-constructed PSO variant, which was obtained by stepwise transformation and simplification of the standard PSO algorithm. Witt conducted a runtime analysis of a special variant of the so-called *guaranteed convergence PSO* [Wit09].

2.5 Multi-objective Particle Swarm Optimization

Many real-world optimization problems require the simultaneous optimization of two or more objectives. For instance, the design of hardware/software systems, which can be found in cars or mobile phones, involves the optimization of many, usually conflicting criteria like monetary cost, data-throughput, or power consumption [BTT98]. Other examples are portfolio optimization (risk vs. expected return) [EKS04], or manufacturing processes (fabrication cost vs. quality). As the optimization goals are usually conflicting, most algorithms search for a set of so-called *trade off* solutions.

Classical approaches for solving multi-objective optimization problems are for instance weighted sum scalarization or the ε-constraint method (see, e.g., [Ehr05]). Additionally, many meta-heuristic approaches were extended for solving multi-objective problems. There exist multi-objective variants for evolutionary algorithms [Deb01, ZT99, ZLT01, DAPM02, ZK04, CVL02], simulated annealing [BSMD08, SK06], ant colony optimization [GMCH07], and particle swarm optimization [RSC06].

2.5.1 Multi-objective Optimization

The task of multi-objective optimization is to simultaneously optimize n' objectives $f_i : \mathcal{S} \rightarrow \mathbb{R}$, $i = 1, \ldots n'$, $n' \geq 2$. It is assumed that each function is to be minimized. Hence, a multi-objective optimization problem can be described by an objective function $f : \mathcal{S} \rightarrow \mathbb{R}^{n'}$, where $\mathcal{S}$ is the n-dimensional search space (or *decision space*) of the problem, and the image $\mathcal{Z} \subseteq \mathbb{R}^{n'}$ of $\mathcal{S}$ under f is the n'-dimensional *objective space*. As PSO is mainly designed for continuous optimization problems, let $\mathcal{S} \subseteq \mathbb{R}^n$ in this section. The situation is depicted in Figure 2.9.

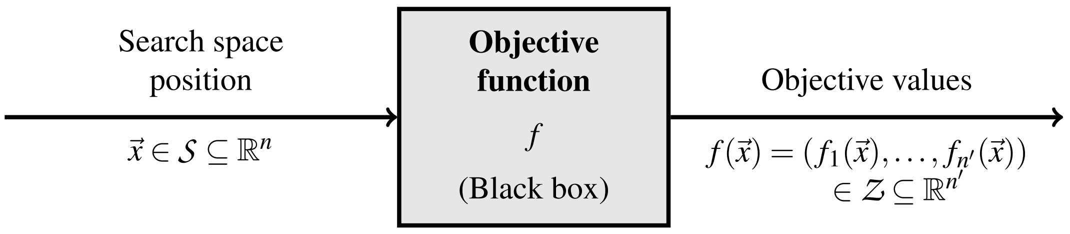

Search space position
$\vec{x} \in \mathcal{S} \subseteq \mathbb{R}^n$
Objective function
f
(Black box)
Objective values
$f(\vec{x}) = (f_1(\vec{x}), \ldots, f_{n'}(\vec{x}))$
$\in \mathcal{Z} \subseteq \mathbb{R}^{n'}$

Figure 2.9: Typical scenario when solving continuous multi-objective optimization problems (based on an illustration in [ZTL+03]).

In contrast to single-objective optimization problems, it is in general not possible to define a total order on $\mathcal{Z} \subseteq \mathbb{R}^{n'}$. Sometimes, user preferences can be determined beforehand, and a multi-objective problem can be transformed into a single-objective one by, e.g., building a weighted sum, minimizing the distance to an ideal solution, or by defining a lexicographic order on the objectives [Deb01, Ehr05]. However, often it is more desirable to present a set of equally good solutions to the decision maker. This is possible due to the fact that a partial order $\preceq \subseteq \mathcal{Z} \times \mathcal{Z}$, denoted as *weak dominance*, can be defined on the objective space (see, e.g., [Lau03]):

$$f(x_1) \preceq f(x_2) :\Leftrightarrow \big(\forall i \in \{1, \ldots, n'\} : f_i(x_1) \leq f_i(x_2)\big) \tag{2.19}$$

with $x_1, x_2 \in \mathcal{S}$. Based on this definition, an order relation $\preceq_f \subseteq \mathcal{S} \times \mathcal{S}$, which depends on the objective function f, can be specified for elements $x_1, x_2 \in \mathcal{S}$:

$$x_1 \preceq_f x_2 :\Leftrightarrow f(x_1) \preceq f(x_2) \tag{2.20}$$

For the sake of brevity, the index f will be skipped in the following. Let $x_1, x_2 \in \mathcal{S}$. The relations $\prec$ (*dominance*), $\sim$ (*equivalence*), $\parallel$ (*incomparability*) $\subseteq \mathcal{S} \times \mathcal{S}$ (see, e.g., [Lau03]) are defined as follows:

$$\begin{array}{llll}
x_1 \prec x_2 & \Leftrightarrow & x_1 \textit{ dominates} \text{ (is preferable to) } x_2 & \Leftrightarrow \quad (x_1 \preceq x_2 \wedge x_2 \not\preceq x_1) \\
& \Leftrightarrow & \multicolumn{2}{l}{(\forall i \in \{1,\ldots,n'\} : f_i(x_1) \leq f_i(x_2) \wedge \exists i \in \{1,\ldots,n'\} : f_i(x_1) < f_i(x_2))} \\
x_1 \sim x_2 & \Leftrightarrow & x_1 \text{ and } x_2 \text{ are } \textit{equivalent} & \Leftrightarrow \quad (x_1 \preceq x_2 \wedge x_2 \preceq x_1) \\
& \Leftrightarrow & (\forall i \in \{1,\ldots,n'\} : f_i(x_1) = f_i(x_2)) & \\
x_1 \parallel x_2 & \Leftrightarrow & x_1 \text{ and } x_2 \text{ are } \textit{incomparable} & \Leftrightarrow \quad (x_1 \not\preceq x_2 \wedge x_2 \not\preceq x_1)
\end{array}$$

These relations are illustrated in Figure 2.10 on a multi-objective optimization problem with the two objective functions f_1 and f_2 which are to be minimized. E.g., assume that $A,\ldots,J$ are available cars, and we want to optimize both the fuel consumption (f_1) and the price (f_2). Obviously, we would prefer car B over car E as it is both cheaper and consumes less gas. Hence, $B \prec E$ holds. We would also prefer B over H. Although both cars have the same monetary cost, car B consumes less gas. In other words, B also dominates H, $B \prec H$. Although I and J are not the same cars, they are equivalent with respect to the optimization goals as they are mapped to the same objective vector: $I \sim J$. However, it is not clear, if car A or car B is to be preferred. B is cheaper than A, but its fuel consumption is higher, whereas A is more expensive but consumes less gas. These solutions are incomporable, or $A \parallel B$.

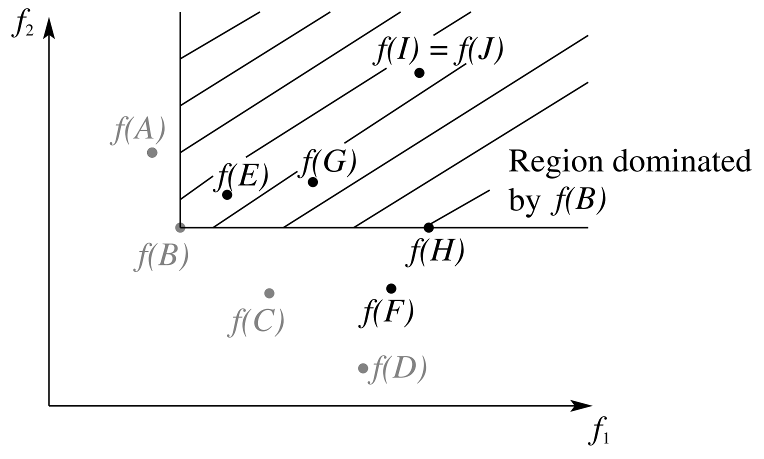


Figure 2.10: Example of the objective space of a multi-objective optimization problem with two objective functions f_1 and f_2 which are to be minimized.

A solution $x^* \in \mathcal{S}$ is called *Pareto optimal* if there exists no solution $x \in \mathcal{S}$ with $x \prec x^*$. The set $\mathcal{P}_\mathcal{S} \subseteq \mathcal{S}$ of all Pareto optimal solutions is called *Pareto optimal set*, whereas the set $\mathcal{P}_\mathcal{Z} \subseteq \mathcal{Z}$ of the corresponding objective vectors is denoted as *Pareto optimal front*.

When solving multi-objective optimization problems, usually the goal is to present a set of incomparable solutions to the decision maker who has to choose among them according to his/her preferences. A set $\mathcal{A} \subseteq \mathcal{Z}$ which contains only incomparable solutions is called *approximation set* [ZTL+03]. In Figure 2.10, the sets $\mathcal{A}_1 = \{f(A), f(B), f(C), f(D)\}$ and $\mathcal{A}_2 = \{f(D), f(E)\}$ are examples of approximation sets.

Most meta-heuristic optimization algorithms do not guarantee to find the Pareto optimal solutions. Instead, their output is an approximation of the true Pareto front, typically an approximation set. In order to compare the performance of different multi-objective optimizers, the following steps are usually carried out in experimental studies:

1. Choose a suitable set of benchmark functions, e.g. from the ZDT functions [ZDT00], the DTLZ problems [DTLZ02], the WFG benchmarks [HHBW06], the benchmarks of Okabe et al. [OJOS04], and/or real world applications.

2. Perform multiple runs per algorithm and benchmark.

3. Compare the performance of different algorithms by visualizing the achieved approximation sets and by computing commonly-used quality meassures such as hypervolume [ZT98, ZTL+03], binary ε-indicator [ZTL+03], inverted generational distance [SC05], spacing [VL00], and coverage [ZT98], to assess both to what extent the algorithms approached the true Pareto front, and the spread and diversity of the obtained solutions. A thorough study on the capabilities and restrictions of quality indicators was carried out by Zitzler et al. [ZTL+03].

2.5.2 PSO for Multi-objective Problems

When adapting particle swarm optimization for multi-objective problems, the most crucial question is how to choose a particle's private and local guide. As often a total order cannot be defined on the objective space, it is not possible to directly adopt the concepts of single-objective particle swarm optimization. Most multi-objective PSO algorithms maintain an external repository to store (a subset of) the best solutions found so far by the swarm. A particle's local guide is then chosen among them according to a specified strategy.

External repositories for storing good solutions, also denoted as *archives*, are also frequently used in multi-objective evolutionary algorithms. They can either be unbounded in size by means of efficient data structures such as dominated and non-dominated trees [FES03], or their size can be restricted by using, for instance, the adaptive grid archive [KC03] or the ε-dominance concept [LTDZ02].

Archives usually contain a set of incomparable solutions, which means first, that a solution can only be added if it is not dominated by any archive member, and second, that whenever a solution is added, all archive members that are dominated by this solution are deleted from the repository.

Selection of Local Guides

In early multi-objective PSO algorithms, the local guides were not selected from an external repository. Instead, in Moore's and Chapman's [MC99] approach, the non-dominated solutions found by each particle are stored in a private repository. Similar to single-objective particle swarm optimization, a social network is defined upon the particles, and a particle's local guide is chosen among all mutually non-dominated solutions of its neighbors.

Most other multi-objective particle swarm optimizers gave up the idea of a social network. Instead, information obtained by the swarm is globally available for any particle. In the approaches of Hu and Eberhart [HE02a] and Parsopoulos and Vrahatis [PV02a], a particle's local guide is selected by explicitely taking the fact that more than one objective is to be optimized into account. For problems with two objectives, Hu and Eberhart redefined neighborhood such that each particle *i* has *k* neighbors, where *k* is a user-specified parameter. The neighbors are dynamically determined in every iteration as those particles which are closest to particle *i* considering the first objective. From these neighbors, particle *i*'s local guide is selected as the one which is best according to the second optimization goal. Parsopoulos and Vrahatis also explained their approach for two-dimensional problems. In their algorithm, two subswarms are used to perform the optimization task. The fitnesses of the individuals in the first subswarm are evaluated according to the first objective function. However, for their velocity update, the best individual in the second swarm is used as their local guide, and vice versa.

In the following, selected multi-objective PSO algorithms in which local guides are chosen from an external repository of non-dominated solutions are briefly decribed, in order to give an impression of how PSO can be extended for multi-objective problems. However, it has to be noted that the list is by far not complete. A detailed overview on multi-objective PSO algorithms can be found in [RSC06].

In the approach of Coello et al. [Coe02a, CPL04], the objective space explored so far is split into equal-sized hypercubes. To each hypercube a fitness value is assigned which is indirectly proportional to the number of archive members in the respective hypercube. In order to select a particle's local guide, first a hypercube is chosen based the fitness values. The higher the fitness, i.e., the less populated the respective hypercube is, the higher is the probability that it is selected. Afterwards, the particle's local guide is chosen uniformly at random from the archive members in the selected hypercube. The approach is illustrated in Figure 2.11 (a).

In contrast to the multi-objective PSO algorithm of Coello et al., the *Sigma* method of Mostaghim and Teich [MT03] as well as the strategies of Alvarez-Benitez et al. [ABEF05] take the particle positions in the objective space into account when selecting local guides. In Mostaghim's and Teich's *Sigma* method a so-called *sigma vector* is assigned to each element of the objective space such that any two points that define a line through the origin have the same sigma values (see the four lines in Figure 2.11 (b)). Each particle chooses the closest archive member as its local guide, where the Euclidean distances of the sigma vectors are used as distance meassure. Alvarez-Benitez et al. allow only archive members which dominate particle i to be selected as particle i's local guide. They introduced three methodologies, from which two are depicted and described in Figures 2.11 (c) and 2.11 (d).

Selection of Private Guides

For selecting a particle's private guide, two conceptually different methods are currently available in the literature:

- Each particle i has a single private guide $\vec{p}_{i,t-1}$ at time step t. The private guide is updated according to one of the following strategies:
    1. It is replaced with the current particle position $\vec{x}_{i,t}$ only if $\vec{x}_{i,t} \prec \vec{p}_{i,t-1}$ [HE02a, MT03].
    2. It is replaced with the current particle position $\vec{x}_{i,t}$ if either $\vec{x}_{i,t} \preceq \vec{p}_{i,t-1}$ or $\vec{x}_{i,t} \parallel \vec{p}_{i,t-1}$ [ABEF05].
    3. As a compromise of these two approaches, the private guide can be set to the prefered solution of $\vec{x}_{i,t}$ and $\vec{p}_{i,t-1}$, if existing. However, if they are incomparable, $\vec{p}_{i,t}$ is chosen uniformly at random among $\vec{p}_{i,t-1}$ and $\vec{x}_{i,t}$ [Coe02a, CPL04].
- The second approach is to maintain a repository for each particle to store its best found solutions. In each time step, the private guides are chosen from these personal archives, e.g., uniformly at random [MC99, FS02, BM06].

2.6 Other Meta-heuristic Optimization Algorithms

The following meta-heuristic optimization approaches belong, like particle swarm optimization, to the class of stochastic search algorithms: Genetic algorithms [Hol62, Hol75, Gol89], evolution strategies [Rec65, Rec73, Sch75, Rec94, BS02], simulated annealing [KGV83, vLA87], differential evolution [SP97, PSL05], and ant colony optimization [DMC91, CDG99, DS04].

In their *no free lunch theorems*, Wolpert and Macready [WM97] proved that, if averaged over all possible optimization problems, all black box optimization algorithms perform equally well. Although it has to be noted that these theorems are based on some assumptions and are therefore not always applicable [AT07, CK03], the results of Wolpert and Macready imply that each black box optimization algorithm has its

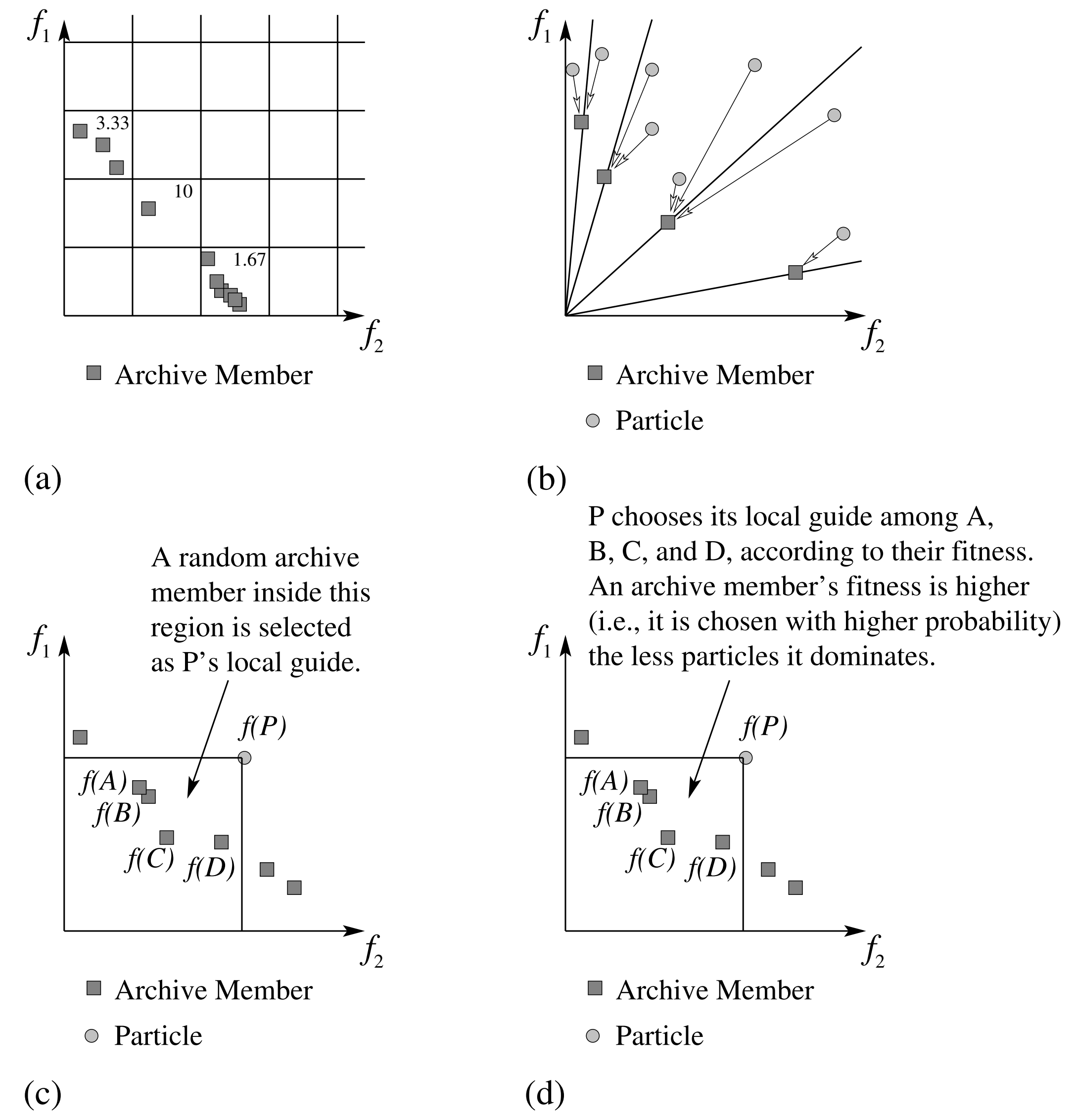

f_1
3.33
10
1.67
f_2
Archive Member
(a)
f_1
f_2
Archive Member
Particle
(b)
A random archive member inside this region is selected as P's local guide.
f_1
f(P)
f(A)
f(B)
f(C)
f(D)
f_2
Archive Member
Particle
(c)
P chooses its local guide among A, B, C, and D, according to their fitness. An archive member's fitness is higher (i.e., it is chosen with higher probability) the less particles it dominates.
f_1
f(P)
f(A)
f(B)
f(C)
f(D)
f_2
Archive Member
Particle
(d)

Figure 2.11: Strategies for selecting local guides from an external repository in multi-objective particle swarm optimization.
(a) Coello et al. [Coe02a, CPL04]
(b) *Sigma* method of Mostaghim and Teich [MT03]
(c) *Random* method of Alvarez-Benitez et al. [ABEF05]
(d) *Prob* method of Alvarez-Benitez et al. [ABEF05]

specific strengths and weaknesses, which, due to the complexity of most algorithms, are usually determined by experimental evaluation on well-defined benchmark problems.

In this section, evolution strategies and ant colony optimization are briefly described. Evolution strategies are a basis of PSO with velocity adaptation presented in Section 5.3, and ant colony optimization is an example of another optimization algorithm from the field of computational swarm intelligence.

2.6.1 Evolution Strategies

Evolution strategies belong to the class of *evolutionary algorithms*, which perform optimization tasks by simulating natural evolution processes according to Darwin's theory. A *population* of *individuals* explores the parameter space of the objective function to be optimized. New individuals, also referred to as *offspring*, are generated by using problem-specific *crossover (recombination)* and *mutation* operators. The combined crossover and mutation step is also called *variation*. A *selection* operator decides which individuals survive and build the population of the next iteration. Usually, the better an individual's objective value, the higher is the probability that it is selected. Evolutionary algorithms can be divided into four major fields:

- Genetic algorithms [Hol62, Hol75, Gol89]
- Evolution strategies [Rec65, Rec73, Sch75, Rec94, BS02]
- Genetic programming [Koz92]
- Evolutionary programming [Fog62, FOW66]

Genetic programming and evolutionary programming are used for evolving computer programs, while genetic algorithms and evolution strategies are stochastic search algorithms for global optimization. The parameter space $\mathcal{S}$ of the optimization problem to be solved can be an arbitrary set on which crossover and mutation operators are defined.

Evolution strategies not only try to optimize the parameters of the objective function (*object parameters*), but also the algorithmic parameters, referred to as *strategy parameters*. Therefore, the algorithmic parameters are devided into a set of so-called *endogenous strategy parameters*, which are adapted to the optimization process, and fixed *exogenous strategy parameters* [BS02].

In an evolution strategy, a population of μ individuals explores the search space. In each iteration, λ offspring are generated by applying problem-specific recombination and mutation operators. For the recombination process, an additional parameter ρ indicates the number of parents per offspring. After λ offspring were produced, the individuals for the next iteration are selected. Evolution strategies can be described with the following commonly-used notation [Rec94, BS02]:

- $(\mu/\rho+\lambda)$-ES if the individuals that build the population in the subsequent iteration are selected from both current population and offspring.
- $(\mu/\rho,\lambda)$-ES if these individuals are selected from the set of offspring. In this case, it is important to choose $\lambda > \mu$ so that the selection operator is able to guide the search in promising directions (otherwise, all children would be selected).

An individual i at time step t is composed of a position $\vec{x}_{i,t}$, also denoted as *object parameter vector*, the corresponding objective value $f(\vec{x}_{i,t})$, and its endogenous strategy parameters $\vec{s}_{i,t}$.

Adaptation of the strategy parameters can take place in various ways, among them *mutative strategy parameter control* [Rec94], the so-called *1/5-rule of Rechenberg* [Rec73], cumulative and/or derandomized approaches [OGH94], and *covariance matrix adaptation* [HO96, HO01].

Although evolution strategies can be used for any search space on which recombination, mutation, and corresponding strategy parameter adaptation is defined (e.g., binary and combinatorial search spaces [BS02]), they are mostly applied to solve continuous optimization problems. In the following, examples for each algorithmic component like recombination and mutation are outlined, assuming $f : \mathbb{R}^n \to \mathbb{R}$.

Parental selection

In an $(\mu/\rho \overset{+}{,} \lambda)$-ES, ρ parents must be selected for each offspring. They are usually chosen uniformly at random among all population members, independently of the individuals' fitness values [BS02].

Recombination

There exist two standard approaches for recombination in ES [Rec73, BS02].

- *Intermediate recombination:* Let $\mathcal{PAR} = \{(\vec{x}_i, \vec{s}_i, f(\vec{x}_i) \mid i = 1, 2, \ldots, \rho\}$ denote the set of (randomly) selected parents, $\mathcal{A} = \{\vec{a}_i \mid i = 1, 2, \ldots, \rho\}$ the set of vectors to recombine (object or strategy parameters), and $\vec{b}$ the resulting recombinant. Then, the d-th component b_d of $\vec{b}$ is the arithmetic mean of the respective parental components:
$$b_d = \frac{1}{\rho} \sum_{i=1}^{\rho} a_{i,d}$$
where $a_{i,d}$ is the d-th component of vector $\vec{a}_i$.
- *Discrete recombination:* Each component of $\vec{b}$ is equal to the respective component of a randomly selected parent.

An example of both approaches is given in Figure 2.12.

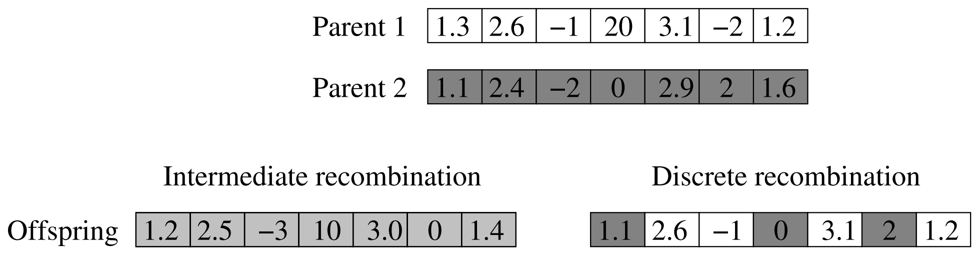


Figure 2.12: Example of two standard evolution strategy recombination approaches: intermediate and discrete recombination.

Mutation of object parameters

Object parameter mutation typically takes place by adding a normally distributed random value with zero mean to the result of the recombination. Let $\vec{y}_{rec}$ be the recombined object parameters, then the offspring's object parameters evaluate to

$$\vec{y} = \vec{y}_{rec} + \vec{z}$$

where $\vec{z}$ can be defined as:

- $\vec{z} = \sigma(N_1(0,1), N_2(0,1), \ldots, N_n(0,1))$
  where $N_k(0,1)$, $k = 1 \ldots n$, are independently drawn from the standard normal distribution [Rec94, BS02]. In this case, there exists only a single endogenous strategy parameter, σ.

- $\vec{z} = (\sigma_1 N_1(0,1), \sigma_2 N_2(0,1), \ldots, \sigma_n N_n(0,1))$
  with n endogenous parameters: $\vec{s} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ [Rec73, BS02].

- In the previous approach only axis-parallel scaling is taken into account. More generally, $\vec{z}$ can be computed as follows [HO96]:
  $\vec{z} = \delta M(N_1(0,1), N_2(0,1), \ldots, N_n(0,1))$
  where $\delta > 0$ is a global step size and $M \in \mathbb{R}^{n \times n}$ is a transformation matrix, which intruduces correlations between the components of $\vec{z}$. Hansen and Ostermeier proposed *covariance matrix adaptation* to adapt the strategy parameters introduced by M. The global step size δ is updated by using a cumulative approach, which takes the mutation steps of more than one iteration into account [HO96, OGH94].

Strategy parameter adaptation

The mutation approaches presented above introduce strategy parameters. In this section, two methodologies for strategy parameter adaptation are briefly described: the $1/5$-rule of Rechenberg [Rec73], and mutative strategy parameter control [Rec94].

The $1/5$*-rule of Rechenberg.* Rechenberg [Rec73] analyzed the (1+1)-ES on two very different functions. He observed that in both cases, the progress rate could be maximized if the probability that an offspring is better than its parent is about $1/5$ during the whole optimization run. If only a single standard deviation is used for all object parameters, the following rule is suggested: The standard deviation is decreased if the success probability (calculated by taking previous iterations into account) drops below the threshold of $1/5$, otherwise it is increased.

Mutative strategy parameter control. In this approach, each individual's strategy parameters are mutated before object parameter mutation is applied. The algorithm is depicted in Algorithm 2.2. The basic idea is that strategy parameters which produced offspring of good quality survive for the next iteration while unsuccessful settings vanish. Mutation of strategy parameters mostly takes place in a multiplicative manner. As an example, if σ is the only strategy parameter, the mutant σ' can be calculated as follows:

$$\sigma' = \sigma \cdot e^{(\tau N(0,1))}$$

where τ is an exogenous *learning parameter*, typically set to $\tau = \frac{1}{\sqrt{n}}$ or $\tau = \frac{1}{\sqrt{2n}}$ [BS02].

Selection

After the offspring was generated by performing recombination and mutation, the individuals which build the population of the next iteration have to be selected. In evolution strategies, selection is usually a deterministic procedure: Only the individuals with best fitness values survive [Rec94,BS02]. There exist two kinds of selection schemes: *Plus selection* selects the next population from both current population and offspring whereas *comma selection* only selects among offspring individuals.

2.6.2 Ant Colony Optimization

Ant Colony Optimization (ACO) [DMC91, CDG99, DS04] is a meta-heuristic optimization approach inspired by the foraging behavior of ants and their capability of finding short(est) paths from their nest to a food source. The ACO algorithm is mostly applied to combinatorial optimization problems. In this section, first the natural basis of ant colony optimization is described. Afterwards, the resulting optimization algorithm is presented by using the *traveling salesperson problem* as an example.

Natural Ants – The Double Bridge Experiment

In 1989, Goss et al. [GAJP89] analyzed the foraging behavior of the Argentine ant *Iridomyrmex humilis*. In their *double bridge experiment*, two paths were offered to an ant colony from their nest to a food source: a shorter and a longer one. After about ten minutes, nearly all ants took the shorter path. Based on previous biologic

Algorithm 2.2
$(\mu/\rho \overset{+}{,} \lambda)$-ES with mutative strategy parameter control (adapted from [BS02])

```
Require: Population size $\mu$, Number of parents per offspring $\rho$, Number of offspring
    per iteration $\lambda$, Fitness function $f$
 1: $t \leftarrow 0$
 2: Initialize population $\mathcal{P}_0 \leftarrow \{(\vec{x}_{i,0}, \vec{s}_{i,0}, f(\vec{x}_{i,0})) \mid i = 1,2,\ldots,\mu\}$
 3: repeat
 4:     for $j = 1,\ldots,\lambda$ do
 5:         $\mathcal{PAR}_{j,t} \leftarrow$ SelectParents($\mathcal{P}_t$, $\rho$)
 6:         $\vec{r}_{j,t} \leftarrow$ RecombinationOfStrategyParameters($\mathcal{PAR}_{j,t}$)
 7:         $\vec{y}_{j,t} \leftarrow$ Recombination($\mathcal{PAR}_{j,t}$)
 8:         $\vec{r}_{j,t} \leftarrow$ MutationOfStrategyParameters($\vec{r}_{j,t}$)
 9:         $\vec{y}_{j,t} \leftarrow$ Mutation($\vec{y}_{j,t}$)
10:     end for
11:     $\mathcal{O}_t \leftarrow \{(\vec{y}_{j,t}, \vec{r}_{j,t}, f(\vec{y}_{j,t})) \mid j = 1,2,\ldots,\lambda\}$
12:     if Comma-selection then
13:         $\mathcal{P}_{t+1} \leftarrow$ SelectPopulation($\mathcal{O}_t$, $\mu$)
14:     else
15:         $\mathcal{P}_{t+1} \leftarrow$ SelectPopulation($\mathcal{P}_t$, $\mathcal{O}_t$, $\mu$)
16:     end if
17:     $t \leftarrow t + 1$
18: until termination criterion is met
```

studies, Goss et al. assumed that the investigated Argentine ant has only little orientation skills. Instead, ants find shortest paths by using indirect communication via pheromone trails, i.e., environmental modifications. They derived a stochastic model for the ants' behavior, which served as a basis for the *ant colony optimization algorithm*.

The double bridge experiment, which led to the stochastic model of the Argentine ants' foraging behavior, is illustrated in Figure 2.6.2. At the beginning, each ant chooses the upper (shorter) or lower (longer) path with equal probability. Hence, in the idealized example each path is chosen by the same number of ants. While moving, ants mark the chosen path with trail pheromones. The more pheromones are placed on a path the more probable other ants will choose the same path. After a while, the ants which have taken the shorter path arrive at the food source. On their way back, they will choose the same path with great probability as there are no pheromones on the other path yet. In the example used in Figure 2.6.2, all ants take the upper branch. In subfigure (d), the other ants finally arrived at the food source. When heading back, there are twice as many pheromones on the shorter path than on the longer one. It is therefore chosen with a probablity of $2/3$ whereas the other branch is only chosen with a probability of $1/3$. Assuming that two ants take the shorter path and only a single ant takes the longer one, the difference in pheromone concentration on the two paths further increases. Eventually, nearly all ants take the shorter branch as a result of this *autocatalytic* or *positive feedback process*.

The Ant Colony Optimization Algorithm

In the following, two representative ACO algorithms, namely *ant system* [DMC91] and *ant colony system* [DG97] are described by using the *traveling salesperson problem (TSP)* as an example application (see page 12 for a description of the TSP).

Ant System (AS), also referred to as *ant-cycle* [DMC91], was one of the first ACO algorithms described in the literature. It is the basis for many improved ACO algorithms such as ant colony system [DG97].

In AS, each ant starts at a randomly chosen vertex, and adds, step-by-step, new nodes until the tour is completed. Similar to natural ant colonies, the artificial ants cooperate via indirect communication using pheromone trails. Let $G = (V, E)$ denote the graph of the traveling salesperson problem. For each edge $(v_i, v_j) \in E$, a pheromone value $\tau_{i,j}$ is maintained, which indicates how desirable it is that an artificial ant chooses this edge for its tour. The pheromone values are updated during the optimization. If an ant k is located at vertex v_i, the probability that it proceeds to vertex v_j is given by the following *random-proportional state transition rule*:

$$p_{i,j}^{k} = \begin{cases} \frac{\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}{\sum_{v_l \in J_k(i)} \left(\tau_{i,l}^{\alpha} \cdot \eta_{i,l}^{\beta} \right)} & \text{if } v_j \in J_k(i) \\ 0 & \text{otherwise} \end{cases} \tag{2.21}$$

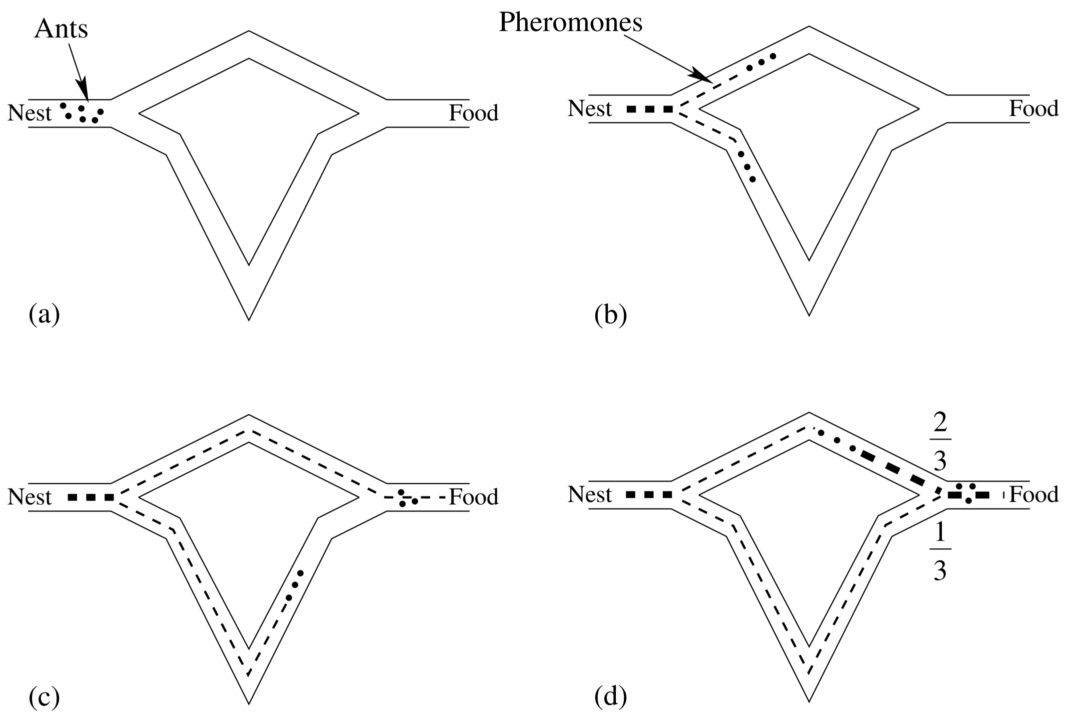


Figure 2.13: The *double bridge experiment* as described by Goss et al. [GAJP89].

where $J_k(i)$ denotes the set of vertices which still have to be visited by ant k located at vertex v_i, $\eta_{i,j}$ is a heuristic value typically set to $1/d_{i,j}$ where $d_{i,j}$ is the distance (weight) from v_i to v_j, and α and β are user-defined parameters, which determine the relative importance of pheromone trail and heuristic information. Standard values for the parameters are $\alpha = 1$ and $\beta \in [2,5]$ [DS04].

Let L_k be the length of the tour constructed by ant k. After each ant has completed its tour, first a certain percentage of all pheromone values evaporates. Then, each ant k increases the pheromone values on the edges it has used. The amount of pheromones added by an ant is indirect proportional to the length of its tour, i.e., ants which found good (short) tours are allowed to deposit more pheromones. The pheromone values are updated according to

$$\tau_{i,j} = (1-\rho)\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} \tag{2.22}$$

where $0 < \rho \leq 1$ is the pheromone evaporation rate and

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{1}{L_k} & \text{if } (v_i, v_j) \in \text{ tour constructed by ant } k \\ 0 & \text{otherwise.} \end{cases}$$

Due to the pheromone update rule, edges which are used in many good tours receive high pheromone values, and the search is guided towards promising directions.

Ant colony system (ACS) [DG97] introduces three major modifications to AS. First, the state transition rule is modified such that with a certain probability q_0 an ant chooses the "best" edge for its next step. This way, the importance of exploitation is emphasized. If ant k is located at v_i, it chooses vertex v_j according to the following *pseudo-random-proportional rule*:

$$v_j = \begin{cases} \arg\max_{v_l \in J_k(i)} \{\tau_{i,l}^{\alpha} \cdot \eta_{i,l}^{\beta}\} & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases}$$

where q is drawn uniformly at random in $[0,1]$, and S is a random variable selected according to the probability distribution given by Equation (2.21).

Second, each time an ant moves from vertex v_i to v_j, $\tau_{i,j}$ is decreased to increase the probability that other ants will explore alternative paths.

Third, for the pheromone update given in Equation (2.22), only the best tour constructed so far is used. This modification increases the convergence speed of the ACO algorithm, which is crucial for large TSP instances.

Ant colony optimization can be used for any kind of optimization problem that allows to construct solutions step-by-step. A more general definition of ACO and other applications such as routing, scheduling and assignment problems are discussed in [DS04].

3. Theoretical Analysis of PSO for Box-constrained Problems

The theoretical analysis of particle swarm optimization has gained increasing interest in recent years in order to understand the swarm dynamics that lead to the success of the PSO algorithm. A detailed review on existing analyses is given in Section 2.4. Based on both deterministic and stochastic models, parameter guidelines were extracted and the behavior of particle swarms, including results about convergence properties, expected runtime on specified benchmarks, and the particles' sampling distribution during stagnation, is more and more understood. All these analyses are based on simplifying assumptions, such as the reduction to a single particle and a one-dimensional problem, non-improving particles (i.e., fixed private and local guides), and unconstrained optimization problems. The theoretical work presented in this chapter (mostly published in [HW07, HW08, HNW09]) complements these studies by providing insights into particle swarm optimization from a completely different perspective. In particular, there are two aspects which, despite their practical relevance, were disregarded in previously analyzed PSO models:

- Constraints within the optimization problem, and
- the peculiarities of high-dimensional parameter spaces.

The task of constrained optimization task is formally defined as (see also Section 2.3):

$$\begin{array}{llll} \text{Minimize} & f(\vec{x}) & & \\ \text{Subject to} & g_i(\vec{x}) \le 0 & i = 1,\dots,m_1 & \textit{(inequality constraints)} \\ & h_j(\vec{x}) = 0 & j = 1,\dots,m_2 & \textit{(equality constraints)} \\ & lb_k \le x_k \le ub_k & k = 1,\dots,n & \textit{(box constraints)} \end{array}$$

The theoretical analysis presented in this chapter assumes *box-constrained* problems as a first step to study PSO algorithms on constrained optimization problems. The optimization task is given by:

$$\begin{array}{llll} \text{Minimize} & f(\vec{x}) & & \\ \text{Subject to} & lb_k \le \vec{x}_k \le ub_k & k = 1,\dots,n & \textit{(box constraints)} \end{array} \tag{3.1}$$

with $f : \mathcal{S} = [lb_1, ub_1] \times [lb_2, ub_2] \times \ldots \times [lb_n, ub_n] \subset \mathbb{R}^n \to \mathbb{R}$. W.l.o.g., $\mathcal{S} = [-r, r]^n$ is assumed. Box constraints can be part of an optimization problem due to one or more of the following reasons:

1. The introduction of box constraints can simplify the problem. If, for instance, one of the parameters is an angle, the number of local optima can be strongly reduced by bounding the respective search space dimension to $[0..2\pi]$.

2. Using search space bounds can avoid needless objective function evaluations. If it is known beforehand that the global optimum and many local ones are located in a specified search space region, the search can be restricted to the relevant part of the underlying parameter space by the introduction of box constraints. This approach was used in the PSO application presented in Chapter 6, and especially pays off when the evaluation of the objective function is very expensive. This is often the case when solving real world problems.

3. Box constraints might be part of the optimization problem.

Moreover, theoretically, any constrained optimization problem can be transformed to a box-constrained problem by defining a homomorphous mapping of the feasible region to a hypercube [KM98, KM99][1]. Hence, the theoretical results derived in this chapter are relevant for a wide range of practical PSO applications.

Existing PSO analyses assume one-dimensional optimization problems due to the fact that the components of a particle's position (resp. velocity) vector are updated independently from one another. However, the implications for high-dimensional optimization problems are not considered. This thesis gives insight into the peculiarities of high-dimensional optimization problems and discusses the consequences for particle swarm optimization. The fact that the geometric properties of high-dimensional spaces are not intuitive and that high-dimensional domains are often hard to tackle, is a well-known mathematical phenomenon, often denoted as *curse of dimensionality* due to a discussion of Bellman [Bel61]. In the field of computer science, the peculiarities of high-dimensional spaces are of particular interest when solving data mining problems such as clustering, classification, nearest neighbor search, and indexing, as these applications often involve high-dimensional data sets [AHK01, VF05, ACXZ05, FWV07, HC09]. The results presented in this thesis show that the curse of dimensionality is an important topic for particle swarm optimization, too, especially when solving problems with box constraints.

In this chapter, it is proven that all particles are initialized very close to the search space boundary when solving high-dimensional optimization problems, and that many particles become infeasible in the first iteration. The treatment of infeasible particles depends on the PSO algorithm's bound handling strategy (see Section 2.3.2). Infeasible particles may, for instance, be reset into the search space according to a specified procedure. As many particles leave the search space in the first iteration, the initial particle swarm behavior strongly depends on the chosen bound handling mechanism.

[1]Note, however, that up to now a suitable mapping cannot trivially be found for optimization problems with disconnected feasible regions. Details can be found in Section 2.3.1. The approach of Koziel and Michalewicz [KM98, KM99] is illustrated in Figure 2.5 on page 27.

Hence, the theoretical results indicate that the bound handling strategy has large impact on particle swarm performance when solving high-dimensional problems. This claim is confirmed by experimental investigations.

Wolpert and Macready have proven that if an optimization algorithm works exceptionally well for one class of optimization problems, its output quality must be poor for another class [WM97]. Therefore, the goal of this thesis is not to identify something like "the best" bound handling strategy, as it depends on the optimization problem which one performs best. Instead, theoretical insights into high-dimensional particle swarm optimization are provided. Some implications for PSO application are presented at the end of this chapter. In the next chapter, the theoretical results are confirmed and extended by a thorough experimental analysis, which explains the effects of different bound handling methods on a variety of commonly-used benchmark functions and reveals their major strengths and weaknesses.

In the following, the symbols Θ and Ω belong to the *big-O notation* for expressing asymptotic behavior [Knu97, p. 108ff]. As particle swarm optimization is a stochastic optimization algorithm, it is often only possible to evaluate the probability of certain events. The following widely-used notion is applied:

Definition 3.1. *A probability $p(n)$ is* exponentially small *in n if there exists a constant $\gamma > 0$ such that $p(n) = e^{-\Omega(n^\gamma)}$. An event $A(n)$ happens* with overwhelming probability (w.o.p.) *with respect to (w.r.t.) n if $P(\text{not}\, A(n))$ is exponentially small in n.*

This means that an overwhelming probability w.r.t. n rapidly approaches 1 when n increases, whilst an exponentially small probability rapidly approaches 0 when n increases.

3.1 The BCPSO Model

In this section, the *box-constrained PSO model (BCPSO model)* is introduced, which is used as basis in the following theoretical analysis. The model comprises both the box-constrained optimization problem and the PSO algorithm. Further variants, denoted as *extended BCPSO model* and *simplified BCPSO model* are defined when needed on pages 60 and 74.

In the BCPSO model, the box-constrained optimization problem is represented by an objective function $f : \mathcal{S} = [-r, r]^n \subset \mathbb{R}^n \to \mathbb{R}$. A minimization problem is assumed. Let $\vec{z}_1, \vec{z}_2 \in \mathcal{S}$. A total order $\leq_f \subseteq \mathcal{S} \times \mathcal{S}$ is defined on $\mathcal{S}$ as:

$$\vec{z}_1 \leq_f \vec{z}_2 :\Leftrightarrow f(\vec{z}_1) \leq f(\vec{z}_2)$$

The PSO algorithm is considered as an iterative stochastic process, similar to the models of Jiang et al. [JLY07a, JLY07b] and Poli et al. [PB07, Pol08]. The original

PSO equations are kept, and repeated here for convenience:

$$\vec{v}_{i,t} = \omega \cdot \vec{v}_{i,t-1} + c_1 \cdot \vec{r}_{1,i,t} \odot (\vec{p}_{i,t-1} - \vec{x}_{i,t-1}) + c_2 \cdot \vec{r}_{2,i,t} \odot (\vec{l}_{i,t-1} - \vec{x}_{i,t-1}) \qquad (3.2)$$
$$\vec{x}_{i,t} = \vec{x}_{i,t-1} + \vec{v}_{i,t} \qquad (3.3)$$

where ω, c_1, and c_2 are constants and t is the iteration counter. The components of $\vec{r}_{1,i,t}$ and $\vec{r}_{2,i,t}$ are independently drawn uniformly at random in $[0,1]$. The components of $\vec{x}_{i,0}$ and $\vec{v}_{i,0}$ are also treated as stochastic variables. They are drawn from random distributions that are specified by the algorithm's initialization strategy. Furthermore, $\vec{p}_{i,0} = \vec{x}_{i,0}$.

An arbitrary neighborhood topology is defined among the particles, i.e., to each particle i a set of indices $\mathcal{I}_{i,t} \subseteq \{1,2,\ldots,m\}$ is assigned, where m is the population size. With the above order relation in mind, $\vec{l}_{i,t}$ is defined as $\vec{l}_{i,t} := \min_{j \in \mathcal{I}_{i,t}}\{\vec{p}_{j,t}\}$. Ties are solved by selecting $\vec{l}_{i,t}$ randomly among the candidates.

After having updated positions and velocities of all particles, the private guides $\vec{p}_{i,t}$ of successful particles are updated, where success can be defined in various ways (see Section 2.1.5).

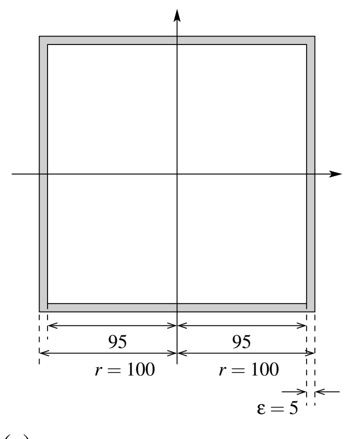
The above definitions completely describe the iterative processes $\{\vec{x}_{i,t}\}$ and $\{\vec{v}_{i,t}\}$ for $i = 1,\ldots,m$ and $t \geq 0$. Note that the components of $\vec{x}_{i,t}$ and $\vec{v}_{i,t}$ are functions of stochastic variables and can therefore be considered as stochastic variables as well.

3.2 Particle Initialization

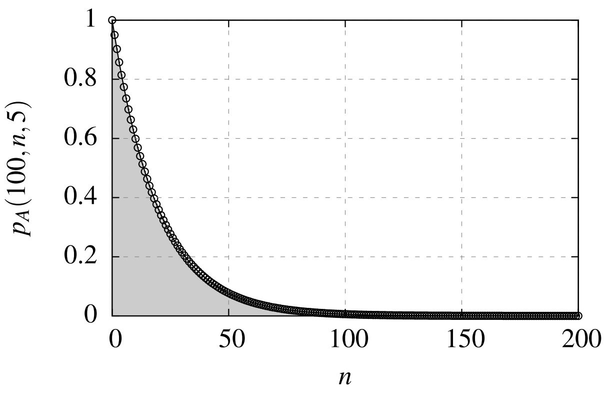
In this section, it is shown that, when solving a high-dimensional problem, each particle is initialized very close to the search space boundary with overwhelming probability. The mathematical proof is simple, however, the fact that particles are initialized very close to the boundary, or, more generally, that most of the search space volume is concentrated in a small shell near the surface, is an important step for understanding high-dimensional particle swarm behavior. It is assumed that particle positions are initialized uniformly at random in $\mathcal{S} = [-r,r]^n$, which is a commonly-used strategy in PSO implementations.

Theorem 3.1. *Consider the BCPSO model, and assume that the components of the initial particle positions $\vec{x}_{i,0}$ are independently drawn uniformly at random from $[-r,r]$ for $i = 1,\ldots,m$. Then, for an arbitrary constant ε, $0 < \varepsilon < r$, the probability $p_A(r,n,\varepsilon)$ that the distance of $\vec{x}_{i,0}$ to the search space boundary is greater than ε is $e^{-\Theta(n)}$.*

Proof. The volume of an n-dimensional hypercube with side length $2r$ is $(2r)^n$. The volume of the inner search space region with more than ε-distance to the boundary is



(a)
Two dimensions: $p_A(100,2,5) \approx 0.9$



(b) $p_A(100,n,5) = \frac{190^n}{200^n}$ for different dimensionalities n

Figure 3.1: The geometric properties of high-dimensional spaces are not intuitive. $p_A(r,n,\varepsilon) = V_{inner\_cube}/V_{cube}$ rapidly approaches zero with increasing dimensionality n.

$(2r-2\varepsilon)^n$. See Figure 3.1a for an illustration. Hence, $p_A(r,n,\varepsilon)$ is given by:

$$p_A(r,n,\varepsilon) = \text{Prob}(\vec{x}_{i,0} \in [-r+\varepsilon, r-\varepsilon]^n) = \frac{(2r-2\varepsilon)^n}{(2r)^n}$$
$$= e^{n \cdot \ln\left(\frac{2r-2\varepsilon}{2r}\right)} = e^{-\Theta(n)} \qquad \square$$

Example 3.2. *Figure 3.1b shows $p_A(r,n,\varepsilon)$ for $r = 100, \varepsilon = 5$, and different problem dimensionalities n.*

This result also explains why it is advantageous to not initialize particles with uniform distribution in some scenarios, as proposed by Richards and Ventura [RV04]: A global optimum which is located near the center of the search space can more easily be found if not all particles are placed near the boundary. However, if nothing is known about the optimization problem beforehand, it makes sense to initialize the particles with uniform distribution as most of the search space volume is located near the boundary.

3.3 Particle Explosion in the First Iteration

Theorem 3.1 made clear that particles are initialized very close to the boundary of a high-dimensional parameter space, with overwhelming probability. In this section, the next step of a PSO algorithm is analyzed, and a formal proof that, w.o.p., many particles become infeasible in the first iteration of a PSO algorithm is derived. The theoretical study focuses on three different velocity initialization strategies. Surprisingly, even initializing velocities to zero cannot prevent that many particles leave the feasible region, which implies that the bound handling strategy strongly affects initial particle swarm behavior. The exact probabilities that a particle becomes infeasible in the first iteration are given in dependence of the velocity initializaton strategy and the algorithmic parameters c_2 and ω. Apart from giving insights into high-dimensional particle swarm optimization, these results also have practical implications, which are discussed in Section 3.6.

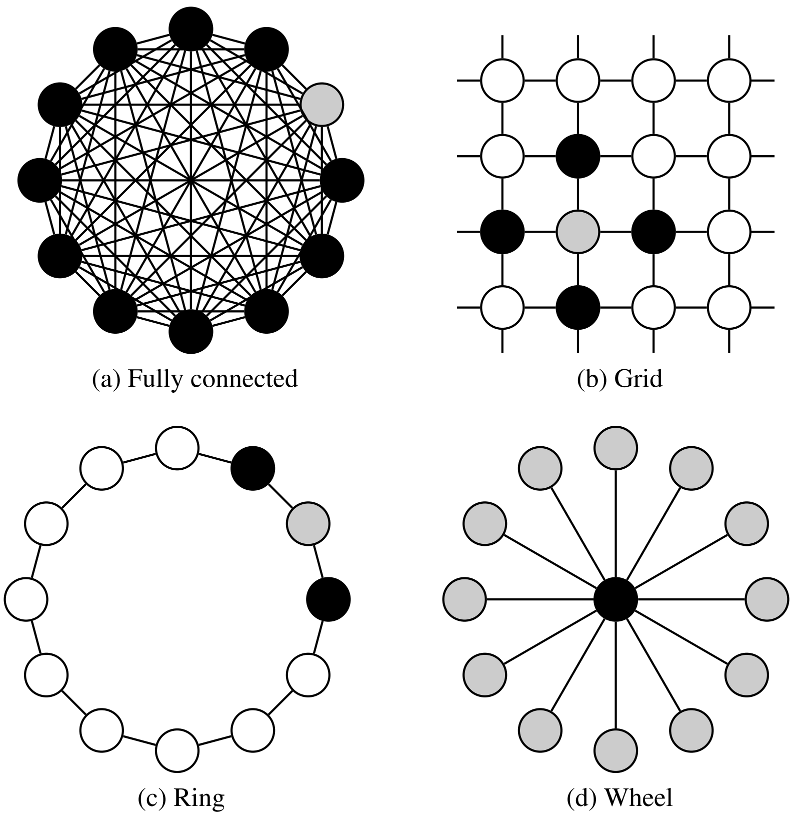
The particle trajectories in the first iteration are analyzed for the three velocity initialization strategies presented in Section 2.1.5 on page 16: uniform, zero, and half-diff initialization. Two main results are derived:

- When using uniform velocity initialization, all particles become infeasible in the first iteration, w.o.p.
- When using zero or half-diff initialization, all particles which are outperformed by at least one neighbor become infeasible in the first iteration, w.o.p., while all others remain feasible.

For many commonly-used neighborhood topologies and realistic optimization problems, i.e., problems which do not include too many flat regions, the majority of the particles is expected to be outperformed by at least one neighbor after the initialization step. W.o.p., the respective particles leave the parameter space in the first iteration. If distinct fitness values are assumed for the particles, the number of individuals which are superior to all their neighbors (and which therefore remain feasible when using zero or half-diff velocity initialization) is bounded by the cardinality of a maximum independent set of the neighborhood graph. Consider the example topologies given in Figure 3.2. In a fully connected swarm, all particles except the best one are outperformed by a neighbor. In the commonly-used ring and grid topologies, still at least half of the particles are outperformed by a neighbor and leave the parameter space in the first iteration. When connecting individuals via the so-called *wheel* [KE01, Ken99] topology, it is possible that only a single particle is outperformed by a neighbor, although it is not very probable. However, the application of this topology often results in slow convergence, as information is delayed by the central node. In an experimental study conducted by Kennedy and Mendes [KM02], the wheel topology was ranked second-last considering 70 distinct social networks and three different performance meassures. The standard PSO algorithm presented by Bratton and Kennedy [BK07] utilizes the ring topology.

For the following analysis, the BCPSO model is extended by the subsequent assumptions. Note that, in accordance with the model, particles are connected via an arbitrary neighborhood topology, e.g., grid, ring, or the fully connected graph.

Assumption 3.1. $1 < c_2 \leq 2$, $0 < \omega \leq 1$, *and* c_2 *and* ω *do not depend on n.*


(a) Fully connected (b) Grid

(c) Ring (d) Wheel

Figure 3.2: For realistic optimization problems, and most commonly-used neighborhood graphs, the majority of the particles is expected to be outperformed by a neighbor after the initialization. Let the gray nodes represent particles that are superior to all their neighbors. Then, when assuming distinct fitness values, the black particles are outperformed by at least one neighbor. They become infeasible in the first iteration, w.o.p., even when initializing velocities to zero or half-diff. When using the wheel topology, it is possible (although not very probable) that only a single particle is outperformed by a neighbor.

Assumption 3.2. *Particles are initialized uniformly at random in the n-dimensional search space $\mathcal{S} = [-r,r]^n \subset \mathbb{R}^n$.*

Assumption 3.3. *For each particle i with $\vec{x}_{i,0} \neq \vec{l}_{i,0}$, $\vec{l}_{i,0}$ is distributed uniformly at random in $\mathcal{S} = [-r,r]^n$.*

Assumptions 3.1 and 3.2 are true for most PSO applications. However, the position of a particle's local guide depends on the optimization problem, even for $t = 0$. The relevance of the theoretical results for PSO implementations is therefore investigated experimentally at the end of Sections 3.3.1, 3.3.2 and 3.3.3. These experiments indicate that Assumption 3.3 is not very restrictive for higher-dimensional problems (cf. Examples 3.5, 3.8, and 3.10).

The extension of the BCPSO model by Assumptions 3.1, 3.2 and 3.3 is denoted as *extended BCPSO model* in the following.

3.3.1 Uniform Velocity Initialization

In this section, it is proven that, w.o.p., all particles become infeasible in the first iteration if velocities are initialized uniformly at random in $\mathcal{S}$. The exact probablities that a particle leaves the search space are computed, in dependence of c_2, ω, and n.

Theorem 3.3. *Consider the extended BCPSO model, and let the components of the initial particle velocities $\vec{v}_{i,0}$ be independently drawn uniformly at random from $[-r,r]$ for $i = 1,\ldots,m$. Then, all particles become infeasible in the first iteration, with overwhelming probability w.r.t. the search space dimensionality n.*

Proof. The proof is divided into two parts:

(i) Each particle i with $\vec{x}_{i,0} = \vec{l}_{i,0}$ becomes infeasible in the first iteration, with overwhelming probability w.r.t. n.

(ii) Each particle i with $\vec{x}_{i,0} \neq \vec{l}_{i,0}$ becomes infeasible in the first iteration, with overwhelming probability w.r.t. n.

Proof of (i). Let particle i be an arbitrary particle with $\vec{x}_{i,0} = \vec{l}_{i,0}$. As $\vec{p}_{i,0} = \vec{l}_{i,0} = \vec{x}_{i,0}$, its position $\vec{x}_{i,1}$ and velocity $\vec{v}_{i,1}$ in the first iteration evaluate to

$$\begin{aligned}
\vec{v}_{i,1} &= \omega \cdot \vec{v}_{i,0} + \underbrace{c_1 \cdot \vec{r}_{1,i,1} \odot (\vec{p}_{i,0} - \vec{x}_{i,0})}_{\vec{0}} + \underbrace{c_2 \cdot \vec{r}_{2,i,1} \odot (\vec{l}_{i,0} - \vec{x}_{i,0})}_{\vec{0}} = \omega \cdot \vec{v}_{i,0} \\
\vec{x}_{i,1} &= \vec{x}_{i,0} + \vec{v}_{i,1} = \vec{x}_{i,0} + \omega \cdot \vec{v}_{i,0} \ .
\end{aligned}$$

Hence, the d-th component of $\vec{x}_{i,1}$ is $x_{i,1,d} = k_1 + k_2$ with $k_1 = x_{i,0,d}$ and $k_2 = \omega \cdot v_{i,0,d}$. The terms k_1 and k_2 are stochastic variables, which are distributed uniformly at random in $[-r, r]$ and $[-\omega r, \omega r]$, respectively, due to the position and velocity initialization strategies. Their density functions are given by

$$
\begin{aligned}
f_{k_1}(z) &= \begin{cases} \frac{1}{2r} & \text{for } -r \leq z \leq r \\ 0 & \text{otherwise} \end{cases} \\
f_{k_2}(z) &= \begin{cases} \frac{1}{2\omega r} & \text{for } -\omega r \leq z \leq \omega r \\ 0 & \text{otherwise} \end{cases} .
\end{aligned}
$$

As k_1 and k_2 are stochastically independent, the density function of their sum $x_{i,1,d}$ is trapezoidal and can be determined by convolution:

$$
\begin{aligned}
f_{x_{i,1,d}}(z) &= \int_{-\infty}^{\infty} f_{k_1}(t) \cdot f_{k_2}(z-t)\mathrm{d}t \\
&= \begin{cases} \frac{1}{4\omega r^2} \cdot z + \frac{1}{4r} + \frac{1}{4\omega r} & \text{for } -r - \omega r < z \leq \omega r - r \\ \frac{1}{2r} & \text{for } \omega r - r < z < r - \omega r \\ -\frac{1}{4\omega r^2} \cdot z + \frac{1}{4r} + \frac{1}{4\omega r} & \text{for } -\omega r + r \leq z < r + \omega r \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{3.4}
$$

Hence, the probability $p_B(\omega)$ that particle i exceeds the search space boundary in dimension d is

$$
p_B(\omega) = \mathrm{Prob}(x_{i,1,d} \notin [-r, r]) = \int_{-r-\omega r}^{-r} f_{x_{i,1,d}}(z) + \int_{r}^{r+\omega r} f_{x_{i,1,d}}(z) = \frac{\omega}{4} .
$$

Figure 3.3 shows a schematic illustration of the trapezoidal density function $f_{x_{i,1,d}}(z)$.

Figure 3.4 shows the relevant part of the probability tree that is obtained for the given situation. The components of $\vec{x}_{i,1}$ are updated independently, and, according to Assumption 3.1, ω does not depend on n. Hence, the probability $p'_B(\omega, n)$ that a particle that satisfies the above assumptions leaves the search space evaluates to

$$
p'_B(\omega, n) = \mathrm{Prob}(\vec{x}_{i,1} \notin [-r, r]^n) = 1 - \left(1 - \frac{\omega}{4}\right)^n = 1 - e^{-\Theta(n)} .
$$

Example 3.4. *Figure 3.5 shows $p'_B(\omega, n)$ for different settings of ω.*

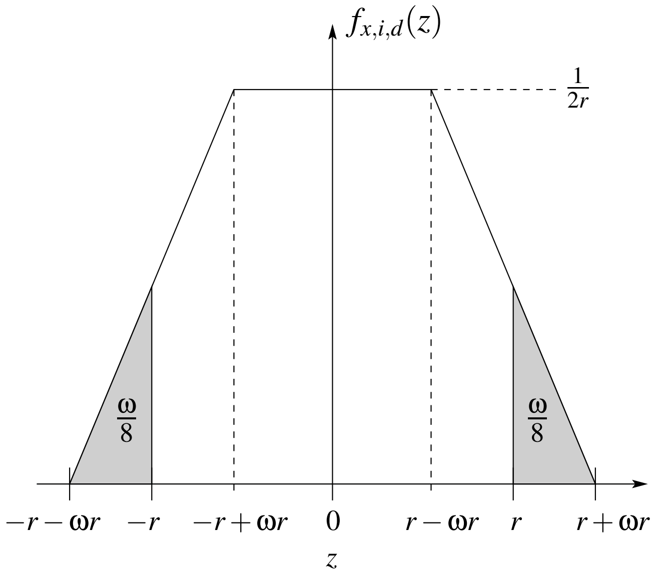


Figure 3.3: Schematic illustration of the density function $f_{x_{i,1,d}}(z)$ as defined in Equation (3.4).

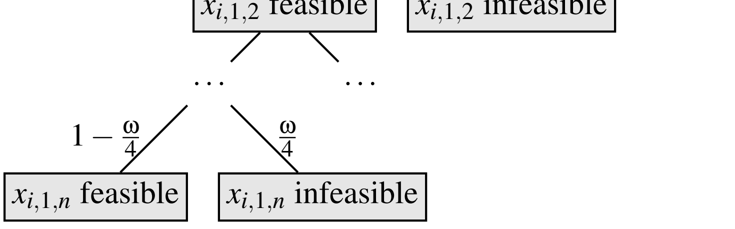


Figure 3.4: Probability tree used in the proof of Theorem 3.3, part (i).

Proof of (ii). Let particle i be an arbitrary particle with $\vec{x}_{i,0} \neq \vec{l}_{i,0}$. As $\vec{p}_{i,0} = \vec{x}_{i,0}$, its position $\vec{x}_{i,1}$ and velocity $\vec{v}_{i,1}$ in the first iteration evaluate to

$$\begin{aligned}\vec{v}_{i,1} &= \omega \cdot \vec{v}_{i,0} + c_2 \cdot \vec{r}_{2,i,1} \odot (\vec{l}_{i,0} - \vec{x}_{i,0})\\ \vec{x}_{i,1} &= \vec{x}_{i,0} + \vec{v}_{i,1} = \vec{x}_{i,0} + \omega \cdot \vec{v}_{i,0} + c_2 \cdot \vec{r}_{2,i,1} \odot (\vec{l}_{i,0} - \vec{x}_{i,0})\\ &= \omega \cdot \vec{v}_{i,0} + (\vec{1} - c_2 \cdot \vec{r}_{2,i,1}) \odot \vec{x}_{i,0} + c_2 \cdot \vec{r}_{2,i,1} \odot \vec{l}_{i,0} \ .\end{aligned} \tag{3.5}$$

The d-th component of $\vec{x}_{i,1}$ can be written as $x_{i,1,d} = k_3 + k_4 + k_5$ with $k_3 = \omega \cdot v_{i,0,d}$, $k_4 = (1 - c_2 r_{2,i,1,d}) \cdot x_{i,0,d}$, and $k_5 = c_2 \cdot r_{2,i,1,d} \cdot l_{i,0,d}$. The terms k_3, k_4, and k_5 are stochastic variables, which are non-identically and independently distributed uniformly at random in respective intervals. Bradley and Gupta derived a formula for the computation of the probability density function of the sum of arbitrary many uniformly distributed random variables [BG02, Theorem 1] (see Appendix A.1). Using their formula and an appropriate computer algebra system, the probability $q_1(r_{2,i,1,d}, c_2, \omega)$ that particle i crosses the search space bound in dimension d, i.e., the probability that $x_{i,1,d} \notin [-r, r]$, is computed to:

$$\begin{aligned} q_1(r_{2,i,1,d}, c_2, \omega) &= \int_{-\infty}^{-r} f_{x_{i,1,d}}(z)\mathrm{d}z + \int_{r}^{\infty} f_{x_{i,1,d}}(z)\mathrm{d}z \\ &= \begin{cases} \frac{-3\omega^2+6c_2 r_{2,i,1,d}\omega-4c_2^2 r_{2,i,1,d}^2}{-12\omega(1-c_2 r_{2,i,1,d})} & \text{if } 0 \leq r_{2,i,1,d} < \frac{\omega}{2c_2} \\ \frac{\omega^2}{24(1-c_2 r_{2,i,1,d})c_2 r_{2,i,1,d}} & \text{if } \frac{\omega}{2c_2} \leq r_{2,i,1,d} < \frac{2-\omega}{2c_2} \\ \frac{4c_2^2 r_{2,i,1,d}^2+6\omega c_2 r_{2,i,1,d}-8c_2 r_{2,i,1,d}+3\omega^2+4-6\omega}{12\omega c_2 r_{2,i,1,d}} & \text{if } \frac{2-\omega}{2c_2} \leq r_{2,i,1,d} < \frac{2+\omega}{2c_2} \\ \frac{24+\omega^2+24c_2^2 r_{2,d}^2-48c_2 r_{2,i,1,d}}{-24c_2 r_{2,i,1,d}(1-c_2 r_{2,i,1,d})} & \text{if } \frac{2+\omega}{2c_2} \leq r_{2,i,1,d} \leq 1 \end{cases} \end{aligned} \tag{3.6}$$

The detailed derivation of Equation (3.6) is presented in Appendix A.2. The probability that particle i becomes infeasible in the first iteration depends on the randomly chosen values $r_{2,i,1,d}$, $d = 1, \dots, n$. According to our model, each $r_{2,i,1,d}$ is distributed uniformly at random in $[0, 1]$. Hence, the probability $p_C(c_2, \omega)$ that a particle violates the boundary in a specific dimension d evaluates to:

$$\begin{aligned} p_C(c_2, \omega) &= \int_0^1 q_1(r_{2,i,1,d}, c_2, \omega)\mathrm{d}r_{2,i,1,d} = \\ &= \begin{cases} (24\omega c_2)^{-1} \cdot (-36\omega+6\omega^2\ln(2)-12\omega^2\ln(2-\omega)+5\omega^2-36\omega\ln(2)+24\omega\ln(2-\omega)+8\ln(2) \\ \quad -16\ln(2-\omega)-3\omega^3\ln(\omega)+2\omega^3\ln(2-\omega)+8\ln(2+\omega)+12\omega\ln(2+\omega)+6\omega^2\ln(2+\omega) \\ \quad -24\omega\ln(c_2)-\omega^3\ln(c_2)+\omega^3\ln(2+\omega)+\omega^3\ln(c_2-1)+24\omega c_2) & \text{if } 2+\omega-2c_2<0 \\ (12\omega c_2)^{-1} \cdot (-10\omega+6\omega^2\ln(2)-6\omega^2\ln(2-\omega)+5\omega^2-12\omega\ln(2) \\ \quad +12\omega\ln(2-\omega)+8\ln(2)-8\ln(2-\omega)-\omega^3\ln(\omega)+\omega^3\ln(2-\omega) \\ \quad +4\ln(c_2)+6-6\omega\ln(c_2)+3\omega^2\ln(c_2)+6\omega c_2+2c_2^2-8c_2) & \text{if } 2+\omega-2c_2\geq 0 \end{cases} \end{aligned} \tag{3.7}$$

The probability $p_C(c_2,\omega)$ solely depends on c_2 and ω, and can be computed for specific settings of these two parameters by using Equation 3.7. In order to prove that particles leave the search space with overwhelming probability, $p_C(c_2,\omega) > 0$ must be shown. Note that $p_C(c_2,\omega)$ does not depend on n. From Equation 3.6, or more generally from the fact that probabilities are always greater than or equal to zero, it follows that $p_C(c_2,\omega)$ can be rewritten as the sum of three terms that are greater than or equal to zero each:

$$p_C(c_2,\omega) = \underbrace{\int_0^{\frac{\omega}{2c_2}} q_1(r_{2,i,1,d},c_2,\omega)\mathrm{d}r_{2,i,1,d}}_{l_1(c_2,\omega)\geq 0} + \underbrace{\int_{\frac{\omega}{2c_2}}^{\frac{2-\omega}{2c_2}} q_1(r_{2,i,1,d},c_2,\omega)\mathrm{d}r_{2,i,1,d}}_{l_2(c_2,\omega)\geq 0}$$
$$+ \underbrace{\int_{\frac{2-\omega}{2c_2}}^{1} q_1(r_{2,i,1,d},c_2,\omega)\mathrm{d}r_{2,i,1,d}}_{l_3(c_2,\omega)\geq 0}$$

For $0 < \omega < 1$, $l_2(c_2,\omega) = \frac{\omega^2\cdot(\ln(2-\omega)-\ln(\omega))}{12c_2} > 0$, and for $\omega = 1$, $l_1(c_2,\omega) = \frac{1+2\ln(2)}{24c_2} > 0$. Hence, $p_C(c_2,\omega) > 0$, and the probability that a particle leaves the n-dimensional search space is

$$p'_C(c_2,\omega,n) = 1-(1-p_C(c_2,\omega))^n = 1-e^{-\Theta(n)} \quad . \tag{3.8}$$

$\square$

Example 3.5. *Part (ii) of the proof makes use of the assumption that the components of each particle's initial local guide are distributed uniformly at random in* $[-r,r]$ *(Assumption 3.3). In order to determine the relevance of the theoretical results for practical particle swarm optimization, the following experiment, denoted as* PSO experiment (PSO Exp.)*, was performed: The PSO algorithm was applied to the CEC 2005 benchmarks f1–f14 [SHL*$^+$*05], excluding f4 and f7 because they are either noisy or unconstrained. More details on the investigated benchmarks can be found in Section 4.2.2. For the PSO algorithm, standard settings as presented in Section 4.1 were used. However, particles are not included in their own neighborhood so that for all particles* $\vec{x}_{i,0} \neq \vec{l}_{i,0}$ *holds, which is an assumption of part (ii) of the proof. For each benchmark and dimensionality,* 10,000 *runs with* 49 *particles were performed. As neighborhood topology, a* 7×7 *grid was used.*

The theoretical results were obtained by pasting Equation (3.7) *into Equation* (3.8).

	Theor. result	*PSO Exp.*
$p'_C(1.496172, 0.72984, 1)$	0.17074	0.15559
$p'_C(1.496172, 0.72984, 30)$	0.99636	0.99585
$p'_C(1.496172, 0.72984, 100)$	0.9999999926	1

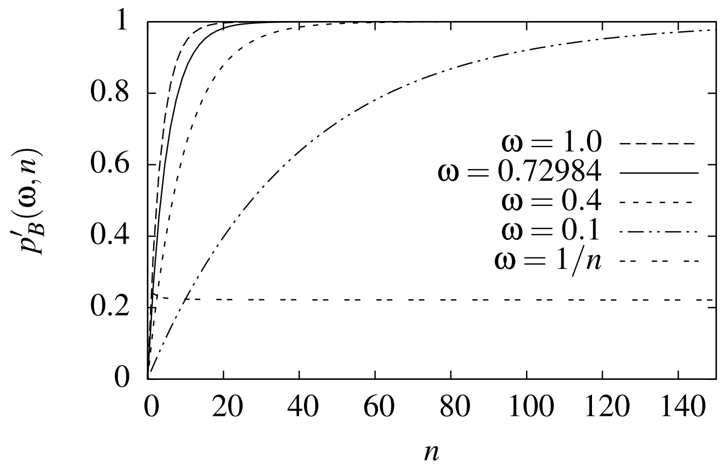


Figure 3.5: Consider particle i that satisfies the assumptions of Theorem 3.3 and for which $\vec{x}_{i,0} = \vec{l}_{i,0}$ holds. The probability $p'_B(\omega, n)$ that particle i becomes infeasible in the first iteration rapidly approaches 1 with increasing dimensionality n, even if ω is set to a small constant value. Note, however, that this is not necessarily the case if the choice of ω depends on n. E.g., $lim_{n\to\infty} p'_B(\omega, n) = c$ for a constant $c < 1$ if $\omega \sim 1/n$.

The only assumption that distinguishes the extended BCPSO model from a real PSO algorithm is Assumption 3.3. In a realistic PSO application, this assumption is not true due to the fact that the determination of a particles' local guide involves function evaluations. Therefore, the distribution of the particles' local guides after initialization depends on the objective function. However, the PSO experiment indicates that Assumption 3.3 is not very restrictive when solving higher-dimensional problems. This observation is also confirmed later in Examples 3.8 and 3.10. From a more general point of view, the PSO experiment confirms part (ii) of the proof: With overwhelming probability, particles become infeasible in the first iteration, if they satisfy the given assumptions.

The probability $p_C(c_2, \omega)$ that particle i with $\vec{x}_{i,0} \neq \vec{l}_{i,0}$ violates a specific boundary in the first iteration depends on c_2 and ω, and is shown in Fig. 3.6. For $c_2 \to 1$ and $\omega \to 0$, $p_C(c_2, \omega)$ approaches zero. However, choosing such small values for c_2 and ω prevents exploration, and can therefore not be recommended.

3.3.2 Zero Velocity Initialization

In this section, it is shown that even initializing particle velocities to zero cannot prevent that many particles become infeasible in the first iteration. To be more precise, a formal proof that all particles with $\vec{x}_{i,0} \neq \vec{l}_{i,0}$ are affected is given. For many commonly-used neighborhood graphs and optimization problems, this is the majority

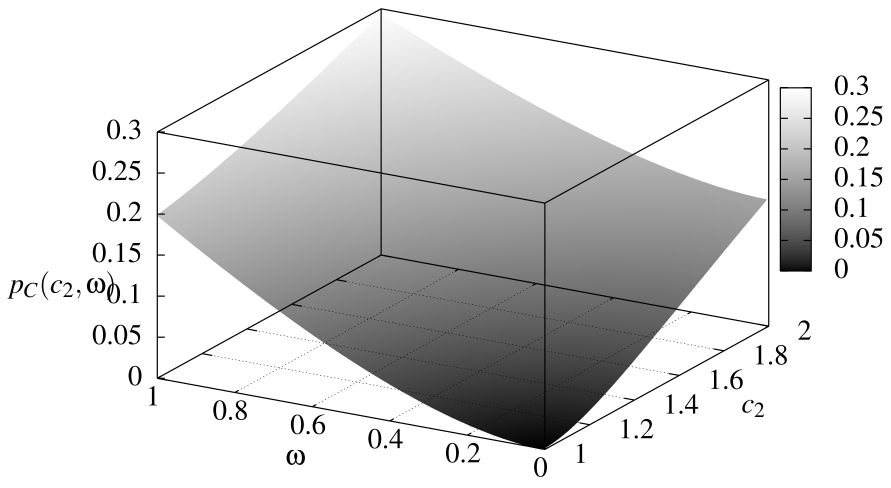


Figure 3.6: The probability $p_C(c_2,\omega)$ that a particle i with $\vec{x}_{i,0} \neq \vec{l}_{i,0}$ violates the boundary in a specific dimension when using uniform velocity initialization.

of the particles, as discussed at the beginning of this section. The following analysis concludes with an example, which shows that particles are slightly less likely to leave the search space in the first iteration compared to the use of uniform velocity initialization.

Theorem 3.6. *Consider the extended BCPSO model, and let the particle velocities be initialized to $\vec{v}_{i,0} = \vec{0}$ for $i = 1,\dots,m$. Then,*

(i) each particle i with $\vec{x}_{i,0} = \vec{l}_{i,0}$ remains feasible in the first iteration, and

(ii) each particle i with $\vec{x}_{i,0} \neq \vec{l}_{i,0}$ becomes infeasible in the first iteration, with overwhelming probability w.r.t. the problem dimensionality n.

Proof of (i). Statement (i) directly follows from $\vec{l}_{i,0} = \vec{p}_{i,0} = \vec{x}_{i,0}$ and $\vec{v}_{i,0} = \vec{0}$:

$$\begin{aligned}\vec{v}_{i,1} =& \underbrace{\omega \cdot \vec{v}_{i,0}}_{\vec{0}} + \underbrace{c_1 \cdot \vec{r}_{1,i,1} \odot (\vec{p}_{i,0} - \vec{x}_{i,0})}_{\vec{0}} + \underbrace{c_2 \cdot \vec{r}_{2,i,1} \odot (\vec{l}_{i,0} - \vec{x}_{i,0})}_{\vec{0}} = \vec{0}\\ \vec{x}_{i,1} =& \vec{x}_{i,0} + \vec{v}_{i,1} = \vec{x}_{i,0} \in \mathcal{S}\end{aligned}$$

Proof of (ii). Let particle i be an arbitrary particle that satisfies the assumptions stated above. Its position and velocity in the first iteration are given by

$$\begin{aligned}\vec{v}_{i,1} &= \underbrace{\omega\cdot\vec{v}_{i,0}}_{\vec{0}} + \underbrace{c_1\cdot\vec{r}_{1,i,1}\odot(\vec{p}_{i,0}-\vec{x}_{i,0})}_{\vec{0}} + c_2\cdot\vec{r}_{2,i,1}\odot(\vec{l}_{i,0}-\vec{x}_{i,0}) = c_2\cdot\vec{r}_{2,i,1}\odot(\vec{l}_{i,0}-\vec{x}_{i,0})\\ \vec{x}_{i,1} &= \vec{x}_{i,0}+\vec{v}_{i,1} = (\vec{1}-c_2\cdot\vec{r}_{2,i,1})\odot\vec{x}_{i,0}+c_2\cdot\vec{r}_{2,i,1}\odot\vec{l}_{i,0}\end{aligned}$$

Hence, the d-th component of the velocity vector computes to

$$x_{i,1,d} = (1-c_2 r_{2,i,1,d})\cdot x_{i,0,d} + c_2 r_{2,i,1,d}\cdot l_{i,0,d} \ .$$

With our assumptions, $x_{i,0,d}$ and $l_{i,0,d}$ are independent random variables, which are distributed uniformly at random in $[-r,r]$. The density function of two non-identically uniformly distributed random variables is trapezoidal. From $r_{2,i,1,d} \leq \frac{1}{c_2}$, it follows $1-c_2 r_{2,i,1,d} \geq 0$, and $x_{i,1,d}$ is trapezoidal distributed in interval $[a_1,b_1]$ with

$$\begin{aligned} a_1 &= (-1+c_2 r_{2,i,1,d})\cdot r - c_2 r_{2,i,1,d}\cdot r = -r\\ b_1 &= (1-c_2 r_{2,i,1,d})\cdot r + c_2 r_{2,i,1,d}\cdot r = r \ .\end{aligned}$$

This means that, if $r_{2,i,1,d} \leq \frac{1}{c_2}$, particle i does not violate the parameter space boundary in dimension d. If $r_{2,i,1,d} > \frac{1}{c_2}$, $x_{i,1,d}$ is rewritten as $x_{i,1,d} = k_6 + k_7$ with

$$k_6 = (1-c_2 r_{2,i,1,d})\cdot x_{i,0,d} \text{ and } k_7 = c_2 r_{2,i,1,d}\cdot l_{i,0,d} \ .$$

Let $r_{2,i,1,d}$ be an arbitrary, but fixed value in $[1/c_2,1]$. Then k_6 and k_7 are distributed uniformly at random in $[(1-c_2 r_{2,i,1,d})\cdot r, (-1+c_2 r_{2,i,1,d})\cdot r]$ and $[-c_2 r_{2,i,1,d}\cdot r, c_2 r_{2,i,1,d}\cdot r]$, respectively. Using Bradley's and Gupta's approach [BG02, Theorem 1], the probability density function $f_{x_{i,t,d}}$ for $x_{i,1,d}$ computes to

$$f_{x_{i,1,d}}(z) = \begin{cases} \dfrac{2z-2r+4rc_2 r_{2,i,1,d}}{-8c_2 r_{2,i,1,d} r^2 + 8c_2^2 r_{2,i,1,d}^2 r^2} & \text{for } (1-2c_2 r_{2,i,1,d})\cdot r \leq z \leq -r\\[2ex] \dfrac{-4r(1-c_2 r_{2,i,1,d})}{-8c_2 r_{2,i,1,d} r^2 + 8c_2^2 r_{2,i,1,d}^2 r^2} & \text{for } -r < z \leq r\\[2ex] \dfrac{-2z-2r+4rc_2 r_{2,i,1,d}}{-8c_2 r_{2,i,1,d} r^2 + 8c_2^2 r_{2,i,1,d}^2 r^2} & \text{for } r < z \leq (-1+2c_2 r_{2,i,1,d})\cdot r\\[2ex] 0 & \text{otherwise} \end{cases}$$

Hence, the probability $q_2(r_{2,i,1,d},c_2)$ that particle i crosses the search space boundary in dimension d is

$$q_2(r_{2,i,1,d},c_2) = \begin{cases} \displaystyle\int_{-\infty}^{-r} f_{x_{i,1,d}}(z)\mathrm{d}z + \int_{r}^{\infty} f_{x_{i,1,d}}(z)\mathrm{d}z = 1-\frac{1}{c_2 r_{2,i,1,d}} & \text{if } r_{2,i,1,d} > \frac{1}{c_2}\\ 0 & \text{otherwise} \ . \end{cases}$$

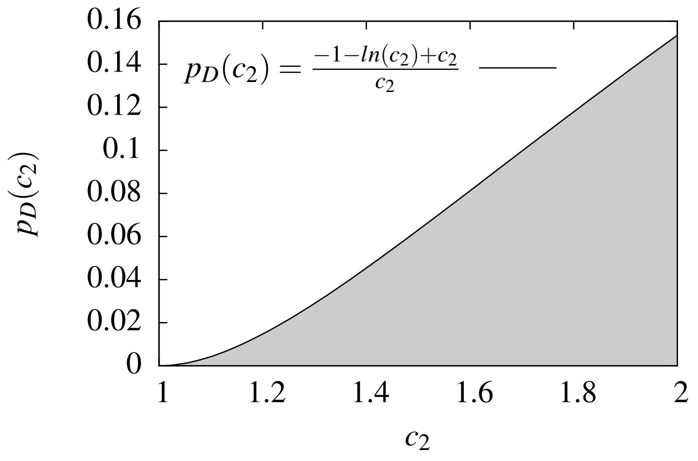


Figure 3.7: The probability $p_D(c_2)$ that a particle that satisfies the assumptions of part (ii) of Theorem 3.6 exceeds the parameter space in a specific dimension is very small for $1 < c_2 \leq 2$. Nevertheless, when solving high-dimensional problems, the probability that particle i becomes infeasible in the first iteration rapidly approaches 1, as for each problem dimension d there is a probability $p_D(c_2) > 0$ that $x_{i,1,d} \notin [-r,r]$. Hence, for large n it is very likely that there exists at least one d with $x_{i,1,d} \notin [-r,r]$.

As $r_{2,i,1,d}$ is distributed uniformly at random in $[0,1]$, the probability $p_D(c_2)$ that a particle violates the search space boundary in a specific dimension is given by

$$p_D(c_2) = \int_0^1 q_2(r_{2,i,1,d}, c_2)\mathrm{d}r_{2,i,1,d} = \frac{-1-\ln(c_2)+c_2}{c_2} \quad . \tag{3.9}$$

The probability $p_D(c_2)$ is shown in Figure 3.7. Inequation $p_D > 0$ holds because $c_2 > 1$ holds. Hence, the probability $p'_D(c_2)$ that a particle which satisfies the given assumptions leaves the n-dimensional search space evaluates to

$$p'_D(c_2,n) = 1-(1-p_D(c_2))^n = 1-e^{-\Theta(n)} \quad . \tag{3.10}$$

□

Example 3.7. *For different settings of c_2, Figure 3.8 shows that $p'_D(c_2,n)$ rapidly approaches 1 with increasing dimensionality n.*

Example 3.8. *For the theoretical results, Equation* (3.9) *was pasted in Equation* (3.10). *The setup of* PSO Exp. *is explained in Example 3.5.*

	Theor. result	*PSO Exp.*
$p'_D(1.496172,1)$	0.06233	0.05176
$p'_D(1.496172,30)$	0.85497	0.83823
$p'_D(1.496172,100)$	0.99840	0.99805

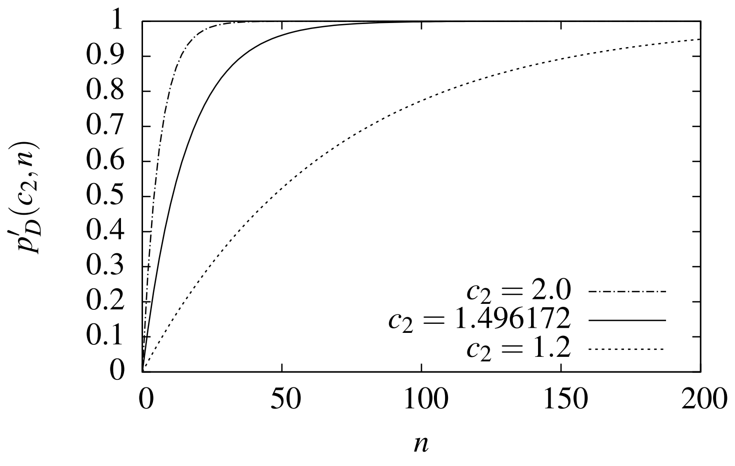


Figure 3.8: The probability $p'_D(c_2,n)$ that a particle that satisfies the assumptions of part (ii) of the proof of 3.6 rapidly approaches 1 with increasing dimensionality n.

The example confirms the relevance of the theoretical results for high-dimensional PSO application. Although the probability that particles leave the search space is smaller than with uniform velocity initialization, it rapidly approaches 1 with increasing problem dimensionality.

3.3.3 Half-diff Velocity Initialization

When using half-diff velocity initialization [C+07], the initial velocities are set to

$$\vec{v}_{i,0} = \frac{1}{2}\left(\vec{z}_i - \vec{x}_{i,0}\right)$$

where $\vec{x}_{i,0}$ is the initial position of particle i, and the vectors $\vec{z}_i$ are independently drawn uniformly at random in $\mathcal{S}$, for $i = 1,\dots,m$. An illustration is given in Section 2.1.5 on page 16.

Half-diff velocity initialization is conceptually different to uniform and zero initialization due to the use of additional search space positions: each particle is attracted by a random solution. As the initial velocities point towards feasible solutions, half-diff initialization might help the particle swarm to stay inside the search space bounds. On the other hand, the two forces of initial velocity and a particle's local guide might accumulate and drive particles beyond the feasible region. The impact of half-diff velocity initialization on the initial swarm behavior is not clear by intuition. The theoretical analysis shows that, again, many particles become infeasible in the first iteration when solving high-dimensional optimization problems. Similar to zero velocity initialization, all particles with $\vec{x}_{i,0} \neq \vec{l}_{i,0}$ are affected, as stated in the following theorem.

Theorem 3.9. *Consider the extended BCPSO model, and let the particle velocities be initialized according to the half-diff strategy as stated above. Then,*

(i) each particle i with $\vec{x}_{i,0} = \vec{l}_{i,0}$ remains feasible in the first iteration, and

(ii) each particle i with $\vec{x}_{i,0} \neq \vec{l}_{i,0}$ becomes infeasible in the first iteration, with overwhelming probability w.r.t. the problem dimensionality n.

Proof of (i). Let particle i be an arbitrary particle that satisfies the given assumptions. As $\vec{p}_{i,0} = \vec{l}_{i,0} = \vec{x}_{i,0}$, its position and velocity in the first iteration evaluate to

$$\begin{aligned}\vec{v}_{i,1} =& \omega \cdot \vec{v}_{i,0} = \tfrac{1}{2} \cdot \omega \cdot (\vec{z}_i - \vec{x}_{i,0})\\ \vec{x}_{i,1} =& \vec{x}_{i,0} + \vec{v}_{i,1} = \left(1 - \frac{\omega}{2}\right) \cdot \vec{x}_{i,0} + \frac{\omega}{2} \cdot \vec{z}_i \in [-r, r]^n\end{aligned}$$

Proof of (ii). Let particle i be an arbitrary particle that satisfies the assumptions stated above. Its position and velocity in the first iteration are given by

$$\begin{aligned}v_{i,1,d} =& \omega \cdot v_{i,0,d} + c_2 \cdot r_{2,i,1,d} \cdot (l_{i,0,d} - x_{i,0,d})\\ x_{i,1,d} =& x_{i,0,d} + v_{i,1,d} = \underbrace{\tfrac{\omega}{2} \cdot z_{i,d}}_{k_8} + \underbrace{c_2 \cdot r_{2,i,1,d} \cdot l_{i,0,d}}_{k_9} + \underbrace{\left(1 - \tfrac{\omega}{2} - c_2 \cdot r_{2,i,1,d}\right) \cdot x_{i,0,d}}_{k_{10}}\end{aligned}$$

for $d = 1, \ldots, n$. Similar to the proof of Theorem 3.3, part (ii), $x_{i,1,d}$ is rewritten as the sum of three stochastic variables, which are distributed uniformly at random in their respective intervals: $x_{i,1,d} = k_8 + k_9 + k_{10}$. Again, the probability density function $f_{x_{i,1,d}}$ of $x_{i,1,d}$ can be computed by using the appraoch of Bradley and Gupta [BG02, Theorem 1]. The probability $q_3(r_{2,i,1,d}, c_2, \omega)$ that particle i violates the d-th search space bound is then computed to (see Appendix A.3 for details):

$$\begin{aligned}q_3(r_{2,i,1,d}, c_2, \omega) =& \int_{-\infty}^{-r} f_{x_{i,1,d}}(z)\mathrm{d}z + \int_{r}^{\infty} f_{x_{i,1,d}}(z)\mathrm{d}z\\ =& \begin{cases} 0 & \text{if } 0 \le r_{2,i,1,d} \le \frac{2-\omega}{2c_2}\\ (6\omega c_2 r_{2,i,1,d}(2-\omega-2c_2 r_{2,i,1,d}))^{-1}(-\omega^3-24c_2 r_{2,i,1,d} & \\ \quad +24c_2^2 r_{2,i,1,d}^2+24\omega c_2 r_{2,i,1,d}+8-8c_2^3 r_{2,i,1,d}^3-12\omega & \\ \quad +6\omega^2-6\omega^2 c_2 r_{2,i,1,d}-12\omega c_2^2 r_{2,i,1,d}^2) & \text{if } \frac{2-\omega}{2c_2} < r_{2,i,1,d} < \frac{1}{c_2}\\ \frac{\omega}{6} & \text{if } r_{2,i,1,d} = \frac{1}{c_2}\\ \frac{-\omega^3+24\omega c_2 r_{2,i,1,d}-12\omega+6\omega^2-6\omega^2 c_2 r_{2,i,1,d}-12\omega c_2^2 r_{2,i,1,d}^2}{6\omega c_2 r_{2,i,1,d}\left(2-\omega-2c_2 r_{2,i,1,d}\right)} & \text{if } \frac{1}{c_2} < r_{2,i,1,d} \le 1\end{cases}\end{aligned} \tag{3.11}$$

According to our model, $r_{2,i,1,d}$ is distributed uniformly at random in $[0,1]$. Hence, the probability $p_E(c_2,\omega)$ that a particle leaves the search space in a specific dimension d is

$$p_E(c_2,\omega) = \int_0^1 q_3(r_{2,i,1,d},c_2,\omega)\mathrm{d}r_{2,i,1,d}$$

$$\begin{aligned}= \quad & (12\omega c_2(\omega-2))^{-1}\cdot(32\omega-22\omega^2+3\omega^3-16\ln(2)+24\omega\ln(2)+16\ln(2-\omega) \\ & -24\omega\ln(2-\omega)+24\ln(c_2)\omega-12\omega^2\ln(2)+12\omega^2\ln(2-\omega)-12\ln(c_2)\omega^2+2\omega^3\ln(2) \\ & -2\omega^3\ln(2-\omega)+2\ln(c_2)\omega^3+2\omega^3\ln(\omega)-24\omega c_2+12\omega^2 c_2-2\omega^3\ln(\omega-2+2c_2)) \quad .\end{aligned} \tag{3.12}$$

Due to the fact that the components of a particle's position vector are updated independently from each other, the probability $p'_E(c_2,\omega)$ that particle i leaves the n-dimensional search space in the first iteration is

$$p'_E(c_2,\omega,n) = 1-(1-p_E(c_2,\omega))^n \quad . \tag{3.13}$$

This probability is overwhelming w.r.t. n if $p_E(c_2,\omega) > 0$, as $p_E(c_2,\omega)$ does not depend on n. The probability $p_E(c_2,\omega)$ can be rewritten as

$$\begin{aligned}p_E(c_2,\omega) &= \int_0^1 q_3(r_{2,i,1,d},c_2,\omega)\mathrm{d}r_{2,i,1,d} \\ &= \int_0^R q_3(r_{2,i,1,d},c_2,\omega)\mathrm{d}r_{2,i,1,d} + \int_R^1 q_3(r_{2,i,1,d},c_2,\omega)\mathrm{d}r_{2,i,1,d}\end{aligned}$$

where R is an arbitrary value in $[0,1]$. As illustrated in Figure 3.9, it is sufficient to show the following three properties in order to prove that the probability $p_E(c_2,\omega)$ is strictly greater than zero:

(a) The function $q_3(r_{2,i,1,d},c_2,\omega)$ is continuous for $r_{2,i,1,d} \in \left(\frac{2-\omega}{2c_2},1\right]$, assuming that c_2 and ω are arbitrary constants that comply with Assumption 3.1[2],

(b) $\exists R, \frac{2-\omega}{2c_2} < R < 1 : q_3(R,c_2,\omega) > 0$, and

(c) $q_3(r_{2,i,1,d},c_2,\omega) \geq 0 \; \forall \; r_{2,i,1,d} \in [0,1]$.

Proof of property (a). From Equation (3.11), it is clear that the function $q_3(r_{2,i,1,d},c_2,\omega)$ is continuous for

(1) $\frac{2-\omega}{2c_2} < r_{2,i,1,d} < \frac{1}{c_2}$,

(2) $\frac{1}{c_2} < r_{2,i,1,d} \leq 1 \quad .$

[2]Assumption 3.1 specifies that $1 < c_2 \leq 2$ and $0 < \omega \leq 1$ hold, and that c_2 and ω do not depend on n.

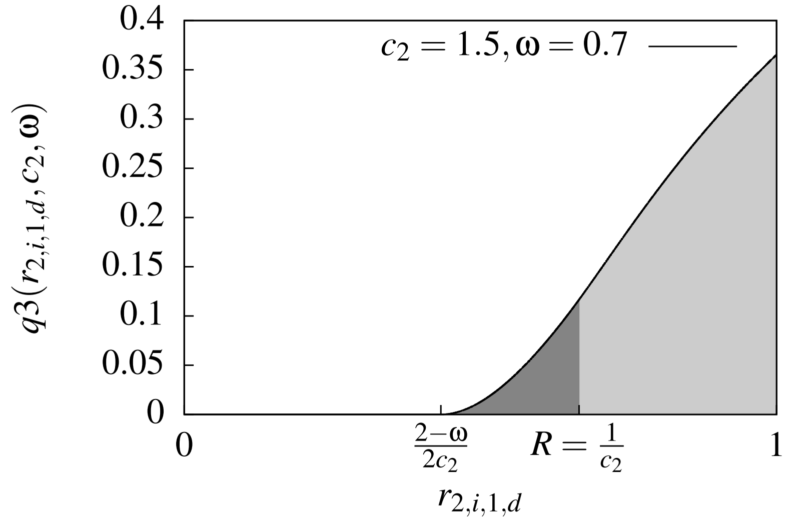


Figure 3.9: Illustration of probability $p_E(c_2, \omega) = \int_0^1 q_3(r_{2,i,1,d}, c_2, \omega) \mathrm{d}r_{2,i,1,d}$. The function $q_3(r_{2,i,1,d}, c_2, \omega)$ is presented in Equation (3.11), and shown for $r_{2,i,1,d} \in [0,1]$. The gray area below $q_3(r_{2,i,1,d}, c_2, \omega)$ corresponds to $p_E(c_2, \omega)$. If the three properties (a)–(c) are fulfilled, this area must be strictly greater than zero.

Furthermore, it can be shown that $q_3(r_{2,i,1,d}, c_2, \omega)$ is continuous at $r_{2,i,1,d} = 1/c_2$, too, by computing the respective left- and right-hand limits. See Appendix A.4 for details.

Proof of property (b). For $R = 1/c_2$, we obtain:

- $R \in \left(\frac{2-\omega}{2c_2}, 1\right)$ due to Assumption 3.1, and
- $q_3(R, c_2, \omega) = \omega/6 > 0$.

Proof of property (c). As $q_3(r_{2,i,1,d}, c_2, \omega)$ is a probability, which was obtained by the integration of density functions (see Equation (3.11)), $0 \leq q_3(r_{2,i,1,d}, c_2, \omega) \leq 1$ holds for the given assumptions.

□

The probability $p_E(c_2, \omega)$ that particle i with $\vec{x}_{i,0} \neq \vec{l}_{i,0}$ exceeds the search space boundary in a specified dimension in the first iteration is plotted in Figure 3.10 for the relevant ranges of c_2 and ω. It approaches zero for $\omega \to 0$ and $c_2 \to 1$. However, as already mentioned during the analysis of uniform velocity initialization in

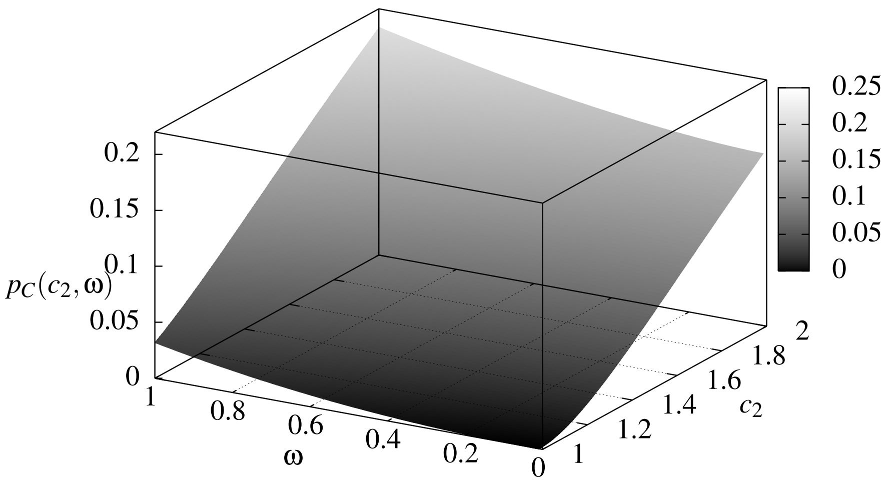


Figure 3.10: The probability $p_E(c_2, \omega)$ that a particle i with $\vec{x}_{i,0} \neq \vec{l}_{i,0}$ violates the boundary in a specific dimension when using half-diff velocity initialization.

Section 3.3.1, setting ω and c_2 to such small values prevents exploration, and can therefore not be recommended.

Example 3.10. *For the theoretical results, Equation* (3.12) *was pasted into Equation* (3.13). *The experimental setup is explained in Example 3.5.*

	Theor. result	*PSO Exp.*
$p'_E(1.496172, 0.72984, 1)$	0.094572	0.083143
$p'_E(1.496172, 0.72984, 30)$	0.949229	0.943357
$p'_E(1.496172, 0.72984, 100)$	0.999952	0.999947

Again, the PSO experiment shows that the theoretical results are relevant for practical PSO application, in particular when solving high-dimensional problems. A comparison with Example 3.5 and Example 3.8 yields that the probability that a particle becomes infeasible settles between the respective probabilities for uniform und zero velocity initialization.

3.4 The Impact of Particle Velocities

In the previous section, initial particle swarm behavior was studied, assuming high-dimensional box-constrained optimization problems. The exact probability that a particle becomes infeasible was computed for different velocity initialization strategies. These probabilities depend on the algorithmic parameters ω and c_2. In some scenarios, if ω and c_2 are selected w.r.t. the problem dimensionality n, the probability that a particle leaves the initialization space approaches a constant $c < 1$ (see Figure 3.5).

In this section, this topic is approached from a more general point of view, and the impact of particle velocities on the particles' tendency to leave the parameter space is studied. The theoretical results presented in this section suggest that an adjustment of the particles' velocities to the problem's dimensionality can help particles to stay inside the feasible region. In particle swarm optimization, the position update equation of particle i is given by

$$\vec{x}_{i,t} = \vec{x}_{i,t-1} + \vec{v}_{i,t} \quad .$$

As the optimization problem is not restricted to a certain class of problems, the characteristics of the probability density functions for $\vec{x}_{i,t-1}$ and $\vec{v}_{i,t}$ are unknown. The following simplifying assumptions are used:

Assumption 3.4. *The components of $\vec{x}_{i,t-1}$ are independently drawn uniformly at random in $[-r,r]$.*

Assumption 3.5. *The components of $\vec{v}_{i,t}$ are independently drawn uniformly at random in $[-\frac{r}{s},\frac{r}{s}]$, $s \geq 1$.*

Both assumptions are not true for real PSO applications. However, they allow the computation of the probability that a particle leaves the search space without making any assumptions on the optimization problem. The explorative behavior of a particle swarm is controlled by the newly introduced parameter s.

The extension of the BCPSO model by Assumptions 3.4 and 3.5 is denoted as *simplified BCPSO model* in the following.

Theorem 3.11. *If the simplified BCPSO model is assumed, the probability that an arbitrary particle i becomes infeasible at time step t is*

$$1 - \left(1 - \frac{1}{4s}\right)^n .$$

Proof. Particle i's position at time step t is computed to

$$\vec{x}_{i,t} = \vec{x}_{i,t-1} + \vec{v}_{i,t} \quad .$$

Let $x_{i,t-1,d}$ and $v_{i,t,d}$ be the d-th components of $\vec{x}_{i,t-1}$ and $\vec{v}_{i,t}$, respectively. As they are distributed uniformly at random in the given intervals, their density functions $f_{x_{i,t-1,d}}$

and $f_{v_{i,t,d}}$ are given by

$$f_{x_{i,t-1,d}}(z) = \begin{cases} \frac{1}{2r} & \text{for } -r \le z \le r \\ 0 & \text{otherwise} \end{cases}$$

$$f_{v_{i,t,d}}(z) = \begin{cases} \frac{s}{2r} & \text{for } -\frac{r}{s} \le z \le \frac{r}{s} \\ 0 & \text{otherwise .} \end{cases}$$

Hence, the d-th component of $\vec{x}_{i,t}$, $x_{i,t,d}$, is the sum of two independent uniformly distributed random variables. The density function $f_{x_{i,t,d}}$ of $x_{i,t,d}$ is trapezoidal and can be determined by convolution:

$$\begin{aligned} f_{x_{i,t,d}}(z) &= \int_{-\infty}^{\infty} f_{x_{i,t-1,d}}(t) f_{v_{i,t,d}}(z-t)\mathrm{d}t \\ &= \begin{cases} \int_{-r}^{z+\frac{r}{s}} f_{x_{i,t-1,d}}(t) f_{v_{i,t,d}}(z-t)\mathrm{d}t & \text{for } -r-\frac{r}{s} \le z \le -r+\frac{r}{s} \\ \int_{z-\frac{r}{s}}^{z+\frac{r}{s}} f_{x_{i,t-1,d}}(t) f_{v_{i,t,d}}(z-t)\mathrm{d}t & \text{for } -r+\frac{r}{s} < z < r-\frac{r}{s} \\ \int_{z-\frac{r}{s}}^{r} f_{x_{i,t-1,d}}(t) f_{v_{i,t,d}}(z-t)\mathrm{d}t & \text{for } -\frac{r}{s}+r \le z \le r+\frac{r}{s} \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} \frac{s}{4r^2}z + \frac{s}{4r} + \frac{1}{4r} & \text{for } -r-\frac{r}{s} \le z \le -r+\frac{r}{s} \\ \frac{1}{2r} & \text{for } -r+\frac{r}{s} < z < r-\frac{r}{s} \\ -\frac{s}{4r^2}z + \frac{s}{4r} + \frac{1}{4r} & \text{for } -\frac{r}{s}+r \le z \le r+\frac{r}{s} \\ 0 & \text{otherwise .} \end{cases} \end{aligned}$$

It follows that the probability $p_F(s)$ that particle i leaves the search space in dimension d evaluates to

$$p_F(s) = \int_{-\infty}^{-r} f_{x_{i,t,d}}(z)\mathrm{d}z + \int_{r}^{\infty} f_{x_{i,t,d}}(z)\mathrm{d}z = \frac{1}{4s} \ .$$

As each component of $\vec{x}_{i,t}$ is computed independently, the probability $p_F(s,n)$ that particle i becomes infeasible in iteration t is given by

$$p'_F(s,n) = 1 - \left(1 - \frac{1}{4s}\right)^n \ . \tag{3.14}$$

□

Note that $1-\left(1-\frac{1}{4s}\right)^n \approx 1-e^{-\frac{n}{4s}}$ holds for large n.

Example 3.12.

$$\begin{aligned} p'_F(2,2) &= 0.23438 \\ p'_F(100,100) &= 0.22144 \\ p'_F(1000,1000) &= 0.22122 \end{aligned}$$

Theorem 3.11, shows that the probability that a particle leaves the bounded search space crucially depends on the interval from which the velocities are chosen. For example, if s is a constant, the probability that a particle becomes infeasible is overwhelming with respect to the search space dimensionality n. On the other hand, $s = n$ implies that the probability that a specified particle i exceeds the search space bound approaches a constant $c < 1$ for increasing n:

$$\lim_{n\to\infty}\left(1-\left(1-\frac{1}{4n}\right)^n\right) = 1-e^{-\frac{1}{4}} \approx 0.22119922$$

This theoretical result indicates that restricting the particles' velocities with respect to the search space dimensionality can be beneficial when solving high-dimensional box-constrained optimization problems with particle swarm optimization.

3.5 An Example: The Sphere Function

In the previous sections it was proven that particle positions are initialized very close to the boundary with overwhelming probability w.r.t. the parameter space dimensionality, and that many particles become infeasible in the first iteration. In this section, some implications are discussed by providing a thorough analysis of PSO initialization when solving the well-known Sphere benchmark. The additional theoretical results give further insight into high-dimensional particle swarm optimization, and are confirmed and extended by experimental investigations at the end of this section. The Sphere function is decribed by:

$$f : \mathcal{S} = [-r,r]^n \to \mathbb{R}, \quad f(\vec{x}) = \sum_{d=1}^{n} x_d^2$$

where n is the problem dimensionality.

The following theoretical investiation was motivated by experimental observations. Zhang et al. [ZXB04] mention that there might be premature convergence on the boundary if Nearest position handling (see page 29) is applied. Own experimental investigations confirm this observation, and demonstrate that this kind of premature convergence also occurs when solving problems that are usually considered as rather unchallenging, such as the Sphere benchmark. First analytical steps for explaining these results are presented in the following. A necessary condition for premature convergence on $\vec{z} \in \mathcal{S}$ is the absence of better solutions $\vec{z}' <_f \vec{z}$ at the beginning of the optimization process, as otherwise, the swarm would be attracted by them. As a first step, the particles' expected initial function values and respective standard deviations are computed. It is shown that they can easily be outperformed by boundary solutions. Moreover, it is proven that the probability that a particle's initial function value is better than the best boundary solution is exponentially small in n. This means that with overwhelming probability w.r.t. n, the aforementioned necessary condition for premature convergence on the boundary is fulfilled. As will be clarified later, this result is rather surprising and not intuitive: Almost certainly, the necessary condition is not fulfilled when solving two- or three-dimensional problems.

Some position handling strategies such as Nearest and Shrink (see page 29) reset particles on the search space bounds which might lead to premature convergence if particles arrive on a good position on the search space boundary. When a particle converges on one boundary by optimizing all other dimensions, the objective function value evaluates to $f(\vec{x}) = r^2$ which is much better than the expected initial function value. This is shown in the following theorem:

Theorem 3.13. *Let the Sphere function be the given optimization problem. If the BCPSO model is assumed and the components of the initial particle positions $\vec{x}_{i,0}$ are independently drawn uniformly at random from $[-r,r]$ for $i = 1,\ldots,m$, the expected initial function value of a particle is $\frac{nr^2}{3}$ and its standard deviation is $\frac{2\sqrt{n}r^2}{3\sqrt{5}}$.*

Proof. Let $y = \sum_{d=1}^{n} x_{i,0,d}^2$ be the initial function value of particle i. Then, $x_{i,0,1}$, $x_{i,0,2},\ldots,x_{i,0,n}$ are stochastically independent and uniformly distributed in $[-r,r]$ each. We obtain:

$$E(y) = n \cdot E(x_{i,0,d}^2) = n \int_{-r}^{r} x^2 \frac{1}{2r} \mathrm{d}x = \frac{nr^2}{3}$$

and

$$\sigma_y = \sqrt{\mathrm{Var}(y)} = \sqrt{n(E(x_i^4) - E(x_i^2)^2)} = \frac{2\sqrt{n}r^2}{3\sqrt{5}}$$

□

Example 3.14. *For $n = 100$ and $r = 100$, the particles' expected initial function value and respective standard deviation evaluate to:*

$$\begin{aligned} E(y) &\approx 333\,333 \\ \sigma_y &\approx 29\,814 \end{aligned}$$

Whenever the swarm converges on a solution $\vec{z} \in \mathcal{S}$, this means that no better solution $\vec{z}' <_f \vec{z}$ was found during the whole optimization process, because if there was a better solution, it would have attracted the particle swarm (assuming a connected neighborhood graph). This fact is intuitively clear, however, recently, it was also proven theoretically by Poli and Broomhead [PB07] that the standard deviation of a particle's trajectory can only converge to 0 if, in our notation, $\vec{p}_{i,t} = \vec{l}_{i,t}$. Hence, particles keep on moving at least until their private guide equals their local guide.

The Sphere function has the following property:

$$\forall \vec{z}_1, \vec{z}_2 \in S: \quad |\vec{z}_1| < |\vec{z}_2| \Rightarrow f(\vec{z}_1) < f(\vec{z}_2) \tag{3.15}$$

It is important to note that the results presented below are relevant not only for the Sphere function but for all functions with the above property. The property is illustrated in Figure 3.11a (top). Assume that the square depicts the two-dimensional search space, then all solutions that are located on the boundary of the (gray) circle have the same objective value, which corresponds to the objective value of the best boundary solution. Solutions inside the circle have a better objective value while solutions outside the circle are worse. Hence, it seems to be very unlikely that the aforementioned necessary condition for premature convergence on the boundary is fulfilled, i.e., that no solution with better quality than a boundary solution was found during the optimization. In fact, for the two-dimensional example, we evaluate

$$\frac{\text{Circle area}}{\text{Square area}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4} \approx 0.785$$

which means, that already in the initialization step, 78.5% of the particles are expected to lie inside the circle. However, high-dimensional search spaces are not intuitive:

Theorem 3.15. *Consider the BCPSO model, and assume that the components of the initial particle positions $\vec{x}_{i,0}$ are independently drawn uniformly at random in $[-r, r]$ for $i = 1, \ldots, m$. Then, the probability $p_G(n)$ of a particle to be initialized inside a hypersphere around the origin with radius r is exponentially small in n.*

Proof. Extending an approach of Jägersküpper [Jäg02, Appendix A] who evaluated the hypersurface area of an n-dimensional sphere with radius $r \geq 0$ to

$$\begin{aligned}
V_{surface}(r,n) &= \int_{\beta_n=0}^{\pi} \int_{\beta_{n-1}=0}^{\pi} \cdots \int_{\beta_3=0}^{\pi} \int_{\alpha=0}^{2\pi} (r \sin\beta_n \\
&\quad \ldots \sin\beta_3 \mathrm{d}\alpha)(r \sin\beta_n \ldots \sin\beta_4 \mathrm{d}\beta_3) \ldots (r \sin\beta_n \mathrm{d}\beta_{n-1})(r \mathrm{d}\beta_n) \\
&= r^{n-1} \cdot 2\pi \cdot \prod_{i=1}^{n-2} \int_0^{\pi} (\sin\beta)^i \mathrm{d}\beta \ ,
\end{aligned}$$

the hypervolume of an n-dimensional sphere with radius r for $n \geq 3$ is given by

$$V_{sphere}(r,n) = \int_{R=0}^{r} V_{surface}(R,n) \mathrm{d}R = \frac{1}{n} \cdot r^n \cdot 2\pi \cdot \prod_{i=1}^{n-2} \int_0^{\pi} (\sin\beta)^i \mathrm{d}\beta \ .$$

The volume of an n-dimensional hypercube with side length $2r$ is $V_{cube}(r,n) = (2r)^n$. As $0 \leq \sin\beta \leq 1, \forall\beta \in [0,\pi]$,

$$g(i) = \int_0^{\pi} (\sin\beta)^i \mathrm{d}\beta$$

is a monotonically decreasing function. Furthermore, we evaluate $g(1) = 2$, $g(5) = \frac{16}{15} > 1$ and $g(6) = \frac{5}{16}\pi < 1$. Hence, the probability that a particle position is initialized inside the n-dimensional hypersphere with radius r is

$$\begin{aligned} p_G(n) =& \frac{V_{sphere}(r,n)}{V_{cube}(r,n)} = \frac{\frac{1}{n} r^n 2\pi \prod_{i=1}^{n-2} \int_0^{\pi} (sin\beta)^i \mathrm{d}\beta}{(2r)^n} \\ \leq& \frac{2\pi}{n2^n} \cdot \prod_{i=1}^{5} \int_0^{\pi} (\sin\beta)^i \mathrm{d}\beta \cdot \prod_{i=6}^{n-2} \int_0^{\pi} (\sin\beta)^i \mathrm{d}\beta \\ \leq& \frac{2\pi}{n2^n} \cdot 2^5 \cdot 1 = e^{-\Theta(n)} \quad . \end{aligned}$$

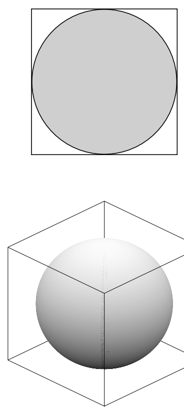
□

Example 3.16. *Figure 3.11 shows $p_G(n)$ for different problem dimensionalities n.*

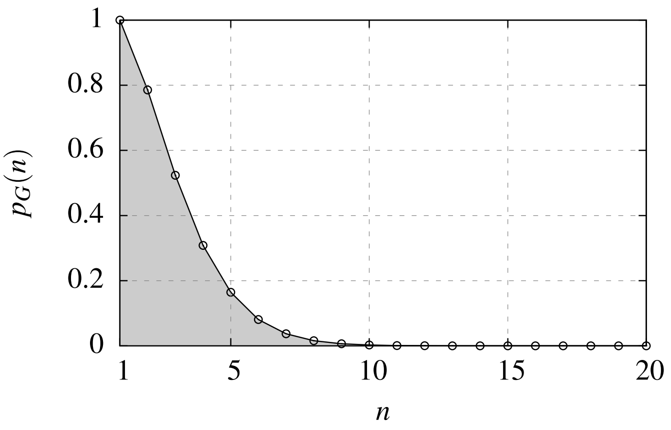
The probability $p_G(n)$ rapidly approaches zero for increasing dimensionality n. For example, for a 30-dimensional problem, $p_G(n)$ computes to $2.04 \cdot 10^{-14}$. This means that in the presence of a high-dimensional parameter space bounded by $[-r,r]^n$, particles are usually initialized outside a sphere around the origin with radius r, which is a necessary condition for the particles to converge on the boundary when solving an arbitrary problem with the property given in Eq. (3.15).

In order to demonstrate the studied effects, a particle swarm optimizer was run on the 2-, 30, 50-, and 100-dimensional Sphere function. 1000 runs per dimensionality were performed. For the PSO algorithm, standard parameters were used: $c_1 = c_2 = 1.49445$ and $\omega = 0.729$ [ES00]. The population size was set to $m = 20$, and the neighborhood topology is a fully connected graph. Particle positions and particle velocities were initialized uniformly at random in $[-r,r]^n$. As bound handling strategy, Nearest position and Unmodified velocity handling (see pages 29 ff.) were applied.

When resetting infeasible particles on the boundary, the swarm might be attracted towards good boundary solutions, and finally converge there, as was already observed by Zhang et al. [ZXB04]. Based on the theoretical study, we expect that this kind of premature convergence occurs more often when the problem dimensionality increases. Figure 3.12 shows the number of runs in which the global optimum was found and the number of runs in which the swarm violated at least one box constraint, in dependence of the search space dimensionality. The higher the problem dimensionality, the more often particles converged on at least one bound. When solving the



(a) $p_G(2) \approx 0.785$, $p_G(3) \approx 0,524$



(b) $p_G(n)$ for different dimensionalities n

Figure 3.11: The geometric properties of high-dimensional spaces are not intuitive: $p_G(n) = V_{sphere}/V_{cube}$ rapidly approaches zero with increasing dimensionality n.

two-dimensional Sphere problem, the global optimum was found in all runs. However, when increasing the dimensionality to 100, only 488 of 1000 optimization runs succeeded to find the global optimum of the Sphere function, whereas in 512 runs the particles converged on at least one parametric bound.

Kennedy and Eberhart propose to initialize positions as well as velocities to random values [KE01, p. 314], but velocities might also be set to zero at the beginning [Eng05, p. 102]. It was shown theoretically in Theorem 3.6 that nevertheless many particles leave the high-dimensional search space with overwhelming probability w.r.t. n in the first iteration. When solving the 30-dimensional Sphere problem with the above PSO algorithm, but by initializing velocities to zero, the global optimum was found in 980 from 1000 runs. If the dimensionality is increased to $n = 100$, the swarm converged on at least one bound in 275 runs. Hence, initializing velocities to zero yielded better results than uniform velocity initialization when solving the Sphere benchmark. However, in accordance with the theoretical analysis, zero velocity initialization cannot always prevent premature convergence on the boundary.

3.6 Consequences for PSO Application

In the previous sections, initial particle swarm behavior was studied in great detail. The theoretical results presented in Sections 3.2 and 3.3 are based on the (extended) BCPSO model, which is very close to a real particle swarm optimizer. The main results are:

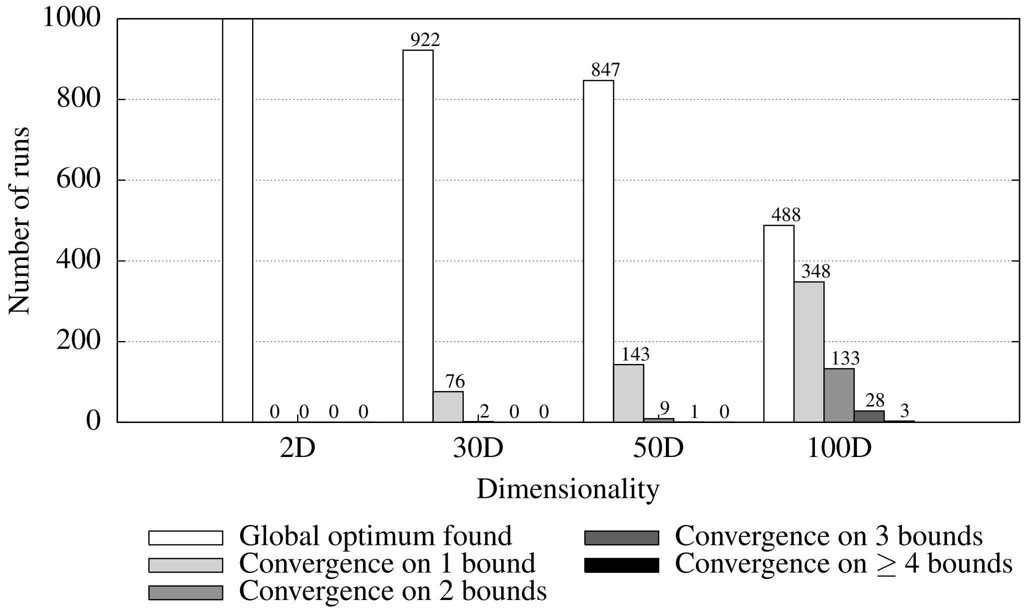


Figure 3.12: Application of a PSO algorithm on the 2-, 30-, 50-, and 100-dimensional Sphere problem. In accordance with the theoretical study, the frequency of premature convergence on the search space boundary increased with the problem dimensionality.

- With overwhelming probability, particles are initialized very close to the boundary (Section 3.2) when solving high-dimensional problems.
- With overwhelming probability, all particles become infeasible in the first iteration when particle velocities are initialized uniformly at random in the parameter space $\mathcal{S}$ (Section 3.3.1).
- With overwhelming probability, all particles that are outperformed by at least one neighbor at the initialization step (i.e., $\vec{x}_{i,0} \neq \vec{l}_{i,0}$)) become infeasible in the first iteration when using zero or half-diff velocity initialization. In many realistic PSO applications, the majority of the particles is affected. All other particles remain feasible (Section 3.3.2 and Section 3.3.3).

In the literature, there exist a lot of strategies to cope with infeasible particles (see Section 2.3). Invalid particles can, for instance, be allowed to explore beyond the search space boundaries, or repaired by moving them to a feasible position. When dealing with box constraints, one can easily construct various repair strategies. From this point of view it is not a problem that particles leave the search space because they can be repaired.

The main conclusion of the theoretical study is the fact that the bound handling strategy is not a rarely used procedure of a PSO algorithm, at least at the beginning of the optimization. Instead, it is applied very often, and therefore strongly affects particle swarm behavior and the final solution quality, in particular, when solving high-dimensional problems. It must be noted that the theoretical analysis only gave thorough results for the first iteration of a PSO algorithm. Although the analysis provides strong evidence that the bound handling procedure essentially influences particle swarm performance, the effect is not proven for the later stages of the algorithm. The conclusions drawn in this section are confirmed by the subsequent extensive experimental investigations.

Due to its importance for particle swarm performance, the bound handling procedure should be chosen with care. As an implication of Wolpert's and Macready's *no free lunch theorems* [WM97], it strongly depends on the optimization problem which bound handling strategy performs best. In real world applications, sometimes a priori knowledge about the optimization problem is available, which can be exploited to choose an adequate bound handling mechanism. The experimental study in the next chapter gives insight into the strenghts and weaknesses of many commonly-used bound handling strategies in order to support the application of PSO algorithms to box-constrained problems.

The theoretical and experimental investigations presented in Section 3.5 showed that in some scenarios, bound handling mechanisms that reset particles on the boundary can lead to premature convergence on the boundary. This phenomenon can occur if good boundary solutions are found at early stages of the algorithm, and was already noticed experimentally by Zhang et al. [ZXB04]. The theoretical analysis can explain this observation: With overwhelming probability, particles are initialized very close to the boundary, and many of them become infeasible in the first iteration. Furthermore, for a special class of problems it was shown that the probability that a particle's initial function value is better than the best boundary solution is exponentially small. As a consequence for practical PSO application, repair strategies that reset particles on the boundary should be avoided. If bound resetting is used, at least the particle velocities in the affected dimensions should be set to zero so that the influence of their private and local guides drives them back into the feasible parameter space.

An important issue when designing bound handling methodologies is to maintain the particles' ability to approach boundary regions. For instance, setting infeasible particles to a random search space position may distract the swarm from boundary regions: Whenever a particle moves slightly too far, it is possibly placed in a completely different search space region. Sending infeasible particles to previously visited positions, e.g., their private guides, may result in slow convergence or even stagnation due to the fact that many particles are infeasible as soon as in the first iteration. Moreover, the theoretical study implies that recomputing a particle's velocity until its position is feasible is not effective when solving high-dimensional problems. The probability that a feasible solution is found by this procedure is exponentially small in the first iteration.

The experimental study in the subsequent chapter suggests that, for instance, reflecting particles at the boundary may be a good choice for high-dimensional particle swarm opimization. However, a lot of different bound handling strategies can be thought of, and their suitability for the current problem at hand has to be analyzed carefully.

The careful selection and well-thought design of bound handling procedures is one possibility to cope with the fact that particles are susceptible to leave the feasible parameter space. Another approach is to eliminate the importance of bound handling for high-dimensional particle swarm optimization. This allows particles to perform their standard movement without being biased by the effects of a bound handling procedure.

Section 3.4 provides first theoretical results in that direction. The analysis, which was conducted on the simplified BCPSO model, indicates that restricting the length of their velocity vectors helps particles to stay inside the feasible region. In particular, if velocities are restricted with respect to the search space dimensionality, e.g., to $[-\frac{r}{n}, \frac{r}{n}]^n$, where $[-r, r]^n \subset \mathbb{R}$ is the underlying parameter space, the importance of bound handling can be strongly reduced. From this theoretical analysis, a particle swarm optimizer with velocity adaptation was derived (see Section 5.3). The use of velocity adaptation diminished the effect of bound handling on particle swarm performance on the investigated benchmark set.

4. Experimental Analysis of PSO for Box-constrained Problems

In this chapter, the results of the theoretical analysis are complemented and confirmed by experimental investigations. For this purpose, commonly-used benchmark functions were used as optimization problems: so-called *traditional benchmarks*, including, for instance, Sphere, Rosenbrock and Rastrigin, and *CEC 2005 benchmarks* [SHL$^+$05]. The PSO algorithm used in this experimental evaluation is based on the standard PSO proposed by Bratton and Kennedy [BK07]. The following issues were studied experimentally:

- *Velocity Initialization*
  In the theoretical study it was proved that none of the investigated velocity initialization methods is able to prevent that, w.o.p., many particles become infeasible in the first iteration. Uniform velocity initialization causes all particles to leave the search space, w.o.p., whereas zero velocity initialization slows down initial exploration. Half-diff velocity initialization seems to have the fewest drawbacks. Nevertheless, the initial particle swarm behavior is similar for all three strategies, and therefore, velocity initialization is expected to have low impact on overall particle swarm performance. This assumption is confirmed by an experimental comparison of the velocity initialization strategies zero, uniform and half-diff.

- *Bound Handling*
  The theoretical study showed that bound handling strongly influences initial particle swarm behavior when solving high-dimensional optimization problems. There exist several strategies to cope with this fact, for instance, the careful design and selection of bound handling mechanisms. This experiment focuses on the following two questions:
  1. Does bound handling significantly influence particle swarm performance, as suggested by the theoretical analysis?
  2. Which are the strengths and weaknesses of commonly-used bound handling methods?

First experimental results on these topics were published in [HW08]. The experimental analysis presented in the following is, however, by far more thorough, by using more bound handling methods and benchmark functions, and by providing a detailed evaluation of the strengths and weaknesses of commonly-used bound handling strategies.

The impact of particle velocities on particle swarm performance is a third important issue to study, considering the results of the theoretical investigation. This topic is postponed to Section 5.3, in which a novel particle swarm optimizer, which was derived from the theoretical insights, is presented: *PSO with velocity adaptation* [HNW09]. The experimental study of PSO with velocity adaptation shows that bound handling has less impact on the achieved solution quality if particle velocities are scaled to a certain length, which is adapted during the optimization. At the same time, PSO with velocity adaptation provides better results than a standard particle swarm optimizer for a wide range of benchmark functions.

4.1 Methodology

The strengths and weaknesses of heuristic algorithms as well as their applicability for certain problems is often analyzed experimentally. The experimental investigations presented in this thesis were carried out according to the following procedure, as described by Barr et al. [BGK$^+$95]:

1. Define the goals of the experiment.
2. Choose measures of performance and factors to explore.
3. Design and execute the experiment.
4. Analyze the data and draw conclusions.
5. Report the experiment's results.

It is important that the goals of an experimental investigation are clearly defined beforehand. This issue is also highlighted in the context of statistics and hypothesis testing (e.g., by Curran-Everett et al. [CETK98]) in order to prevent that regularities, which occur in any data set simply by chance, are misinterpreted. Moreover, a predefined goal has strong impact on the design of an experimental study. The goals of an experiment decide, for instance, about the data that has to be collected and about suitable (statistical) methods for the analysis of the experimental output.

When applied on realistic optimization problems, the most expensive part of a stochastic search algorithm usually is the evaluation of the objective function. Often, time-consuming computations (see, e.g., Chapter 6) or simulations take place. Hence, in the experimental study, the algorithms' performance is meassured in terms of objective value in dependence of the number of executed function evaluations. Two different algorithmic factors are explored: velocity initialization and bound handling.

All parameters besides velocity initialization and bound handling strategy were kept fixed throughout the experimentation. The setup is presented below and summarized in Table 4.1. The experiments were conducted on the testfunctions described in Section 4.2. The problem size is varied in order to investigate the impact of dimensionality on particle swarm optimization, which is a main topic of this thesis.

The data was analyzed by common statistical methods such as the computation of average values, standard deviations, standard errors and confidence intervals, and by means of statistical hyposthesis tests.

4.1.1 Setup

Before presenting the statistical tools that were used for the data analysis, the experimental setup is described in this section. The algorithmic parameters are divided into *variated parameters* and *fixed parameters*.

Setup – Variated Parameters

In the first experiment, the three velocity initialization strategies of the theoretical analysis are considered:

- Zero velocity initialization,
- half-diff velocity initialization, and
- uniform velocity initialization.

From the large number of bound handling strategies that are available in the literature (see, e.g., [Cle06a, ABEF05, ZXB04, RRS04, BK07]), a representative subset had to be chosen for the second experiment. The following strategies were selected. They are decribed in detail in Section 2.3.2.

- *Hyperbolic* [Cle06a], which is a special velocity operator.
- *RandomBack* as described by Clerc [Cle06a]: *Nearest* position handling, and *Invert* velocity handling with z drawn uniformly at random in $[0,1]$.
- *Nearest-Z, Nearest-A, Nearest-U*:
  Nearest position handling and *Zero*/*Adjust*/*Unmodified* velocity handling.
- *Random-Z, Random-A, Random-U*:
  Random position handling and *Zero*/*Adjust*/*Unmodified* velocity handling.
- *Reflect-Z, Reflect-A, Reflect-U*:
  Reflect position handling and *Zero*/*Adjust*/*Unmodified* velocity handling.

- *Infinity* as used in the standard PSO algorithm of Bratton and Kennedy [BK07].
- *Infinity-C*: *Infinity* with velocity clamping. Let $\mathcal{S} = [lb_1, ub_1] \times [lb_2, ub_2] \times \ldots \times [lb_n, ub_n]$ be the search space of the optimization problem. The maximum velocity per component, $V_{max,i}$, is set to $V_{max,i} = (ub_i - lb_i)/2$ for $i = 1, \ldots, n$ (see Section 2.1.5 for details about velocity clamping).

Hyperbolic and RandomBack were found as two good performing methods with quite oppositional characteristics by Clerc [Cle06a]. Nearest, Random, and Reflect are simple and straightforward repair methods with different properties: While Nearest resets particles on the boundary, and is therefore biased towards the boundary (even causing premature convergence on the boundary sometimes [ZXB04, HW07]), Random distracts particles from the boundary [HW07]. Reflect might be a compromise. Infinity was proposed by Bratton and Kennedy as standard method [BK07], and is therefore included in the experimentation. Own preliminary experimental results as well as those of Eberhart and Shi [ES00] suggest that the performance of Infinity can be considerably improved sometimes if velocity clamping is used. Therefore, Infinity-C was regarded as well.

Setup – Fixed Parameters

A standard particle swarm optimizer similar to the one of Bratton and Kennedy [BK07] was used for the experimental investigations. The algorithmic parameters c_1, c_2, and ω were set to $c_1 = c_2 = 1.496172$ and $\omega = 0.72984$ [BK07]. As proposed by Kennedy and Mendes, the swarm is connected via the so-called *von Neumann* topology, a two-dimensional grid with wrap-around edges [KM02], as shown in Figure 2.4 on page 19. A particle is included in its own neighborhood, i.e., each particle has five neighbors (top, bottom, left, right, and itself). Bratton and Kennedy suggest to use 50 particles, however, in this study, the population size was set to $m = 49$ so that the particles can be arranged in a regular 7×7 grid. As opposed to the standard PSO [BK07], all private guides are updated at the same time after the position and velocity updates, in order to simulate particles that act in parallel. If a particle's private guide and its current position evaluate to the same objective value, the private guide is updated to the current position with probability 1/2. If a particle has more than a single best neighbor, its local guide is randomly chosen among the candidates.

For all benchmarks, particles were initialized uniformly at random in the whole search space. Historically, individuals are often initialized in a subspace of the feasible region in order to avoid that the performance of algorithms with center bias is overestimated. E.g., the initialization space of the Sphere function may be set to $[50, 100]^n$ whereas the parameter space is $[-100, 100]^n$ [BK07]. However, in the following experiments, asymmetric initialization ranges are not used due to the following two reasons: First, the CEC 2005 benchmarks do not have their global optimum at the center of the search space, and partly are shifted and rotated versions of standard test problems. Hence, variety is increased if additionally, traditional benchmarks with centered optima are used. Second, when solving real world problems, it is mostly not desirable that particles explore regions beyond the initialization space, which is defined by the box constraints, as solutions outside the search space boundaries are infeasible. In the experimental study, each configuration was repeated 100 times, and each run was terminated after $300,000$ function evaluations.

The velocity initialization strategies were examined by using the bound handling strategies Nearest-Z and Random-Z. In the second experiment, velocities were initialized with half-diff initialization.

Table 4.1: Setup for the experimental analysis.

PSO parameters	
Population size	49
Neighborhood topology	7×7 grid (self included, undirected)
Accelaration coefficients c_1 and c_2	1.496172
Inertia weight ω	0.72984
Craziness (mutation, turbulence)	no
Particle initialization	Uniformly at random in the search space
Variated PSO parameters	
Velocity initialization	Uniform, zero, half-diff
Bound handling	Hyperbolic, RandomBack Nearest-Z, Nearest-A, Nearest-U Random-Z, Random-A, Random-U Reflect-Z, Reflect-A, Reflect-U Infinity, Infinity-C
Experimentation parameters	
Function evaluations per run	$300,000$
Number of runs per configuration	100

4.1.2 Basic Statistical Methods

In statistics, there is a clear distinction between the *statistical population* and a *sample* [Lap90, page 7], [CETK98]. A statistical population is defined as the collection of all possible observations. This could, for instance, be the final solution quality of all possible runs of a particle swarm optimizer with a certain setting. As it is often not possible to obtain the whole population, its parameters such as mean value and standard deviation are estimated based on a *sample*. A sample is a portion of the whole population, which is usually selected randomly.

Let $\{X_1, X_2, \ldots, X_N\}$ be a sample with N observations[1]. The population mean μ is estimated by the *sample mean* $\overline{X}$ [Lap90, CETK98, OM88]:

$$\overline{X} = \frac{\sum_{i=1}^{N} X_i}{N}$$

The population standard deviation σ is estimated by the *sample standard deviation* s [Lap90, CE08, OM88]:

$$s = \sqrt{\frac{\sum_{i=1}^{N} \left(X_i - \overline{X}\right)^2}{N-1}}$$

The *standard error of the mean*, $s_{\overline{X}}$, can be estimated by $s_{\overline{X}} = s/\sqrt{N}$ [CETK98, OM88]. While the sample standard deviation estimates the variability of the population, the standard error is related to the accuracy of the obtained sample mean. The more samples are observed, the more confident one can be that the sample mean reflects the population mean, and the smaller is the resulting standard error.

The standard error of the mean is used to construct the so-called *confidence interval* for the population mean μ. The aim is a statement like *"with a confidence of $p_{ci} = 95\%$, the population mean μ is located in $\left[\overline{X} - a, \overline{X} + a\right]$"*. The actual meaning is the following [Lap90]: If the sampling procedure was repeated, approximately 95% of the constructed confidence intervals would contain the population mean μ. If the population standard deviation σ is unknown, and estimated with the sample standard deviation s instead, the Student t distribution is used for the computation of the interval bounds [Lap90, CETK98]:

$$a = t(\alpha/2, n-1) \cdot s_{\overline{X}}$$

where $\alpha = 1 - p_{ci}$, and $t(\alpha/2, n-1)$ is the $100(1-\alpha/2)$th percentile from a Student t distribution with $n-1$ degrees of freedom. The values for $t(\alpha/2, n-1)$ can be found in respective tables (e.g., [Lap90, Appendix A, Table G]). In the subsequent experiments, sample means of final objective values and 95% confidence intervals are visualized as shown in Figure 4.1. The graphical representation was inspired by Tufte's way of drawing box plots [Tuf07, Chapter 6].

The statistical analysis was carried out by using R [R D08]. For the computation of sample mean, sample standard deviation, and standard error of the mean, the functions `rowMeans`, `sd`, and `se` (library `sciplot`) were utilized, respectively. Confidence intervals were obtained by calling `t.test` with a single sample.

[1]In the following experiments, the sample size is $N = 100$ (number of runs per configuration).

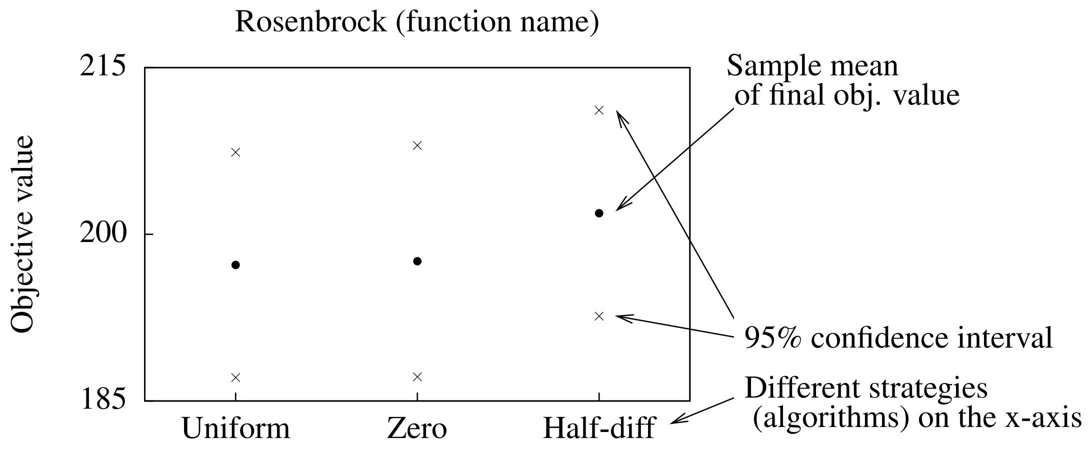


Figure 4.1: Example for a compact representation of sample means and confidence intervals of different strategies (Uniform, Zero, Half-diff).

4.1.3 Statistical Hypothesis Testing

In addition to these standard methods, the population was investigated by statistical hypothesis testing [Lap90, HS06, CE09, OM88, BT78]. There are two hypothesis involved in each test: A *null hypothesis* H_0 and an *alternative hypothesis* H_1. A null hypothesis might, for instance, claim that the mean values of two populations are identical while the alternative hypothesis says the contrary. If the null hypothesis can be rejected due to the results of a hypothesis test, there is statistical evidence that the alternative hypothesis is true. The strength of this claim depends on the so-called *significance level* $\alpha = \text{Prob}(\text{reject } H_0 \mid H_0 \text{ is true})$, which specifies the probability that the null hypothesis is rejected although it is true (*type I error*). Conducting a type I error is usually considered worse than accepting the null hypothesis although it is false, which is referred to as *type II error*. The significance level is typically set to $\alpha = 0.05$ or $\alpha = 0.01$.

The existing hypothesis tests differ in their *test statistic*, a quantity that is computed from the observed data, and used for the decision whether the null hypothesis is rejected or not.

For the subsequent experiments, the so-called *Wilcoxon rank sum test* [Wil45, MW47, HS06, Lap90] was utilized. The aim of this test is to find out if two sample sets A and B were derived from the same population (null hypothesis). For this purpose, a joint set is built from the two sample sets, and ranks are assigned to each observation: the smallest observation gets rank 1, the second-smallest rank 2, and so on. Then, the joint set is split again into the sets A and B. W.l.o.g., the ranks of the members of set A are considered. If, for example, each set A and B

had $N = 3$ members, 20 sets of ranks are possible for the members of sample A: $\{1,2,3\},\{1,2,4\},\dots,\{2,3,4\},\dots,\{4,5,6\}$. If A and B were derived from the same population, each of these rank sets had the same probability of occuring. This fact is the core of the Wilcoxon rank sum test. The test statistic W is the sum of the ranks that belong to the observations in sample set A [Lap90, Wil45]. Obviously, if the null hypothesis is true, very low or very high rank sums are not as probable as intermediate ones: there is, for instance, only a single set of ranks that yields rank sum $(N+1)\cdot N/2$ (the smallest possible rank sum) or $(3N+1)\cdot N/2$ (the largest possible rank sum), but more than one set for intermediate rank sums. For large sample sizes and given that the null hypothesis is true, the distribution of the rank sums approaches the normal distribution [HS06, MW47].

In the one-sided test that is used in the experiments, null and alternative hypothesis are given by (adapted from [HS06]):

$$H_0 : P(X_A < X_B) \le \frac{1}{2} \tag{4.1}$$

$$H_1 : P(X_A < X_B) > \frac{1}{2} \tag{4.2}$$

where X_A and X_B are arbitrary observations from the underlying populations of A and B, respectively. The null hypothesis is rejected if the rank sum of sample set A is extremely low. Let p_r, $r = (N+1)\cdot N/2,\dots,(3N+1)\cdot N/2$, be the probability that rank sum r occurs, given that each ranking has the same probability. The *p-value* is computed to $p = \sum_{i=(N+1)\cdot N/2}^{W} p_i$, and equals the probability that the observed value for the test statistic or a more extreme value is obtained if the null hypothesis was true. The null hypothesis is rejected iff $p < \alpha$. Illustrative examples are given in Appendix B.1.

Note that the Wilcoxon rank sum test is equivalent to the so-called *Mann and Whitney U test* [MW47]. The only difference is a transformation of the test statistic, but the results are identical [Lap90, HS06, OM88]. Ties in the observed data are solved by assigning the mean rank value to the respective observations [Wil45]. However, a correction of the test statistic might be necessary [HS06].

In the experiments, sample set A and sample set B are the final solution qualities of $N = 100$ runs of algorithms A and B. If the null hypothesis is rejected, algorithm A significantly outperformed algorithm B, based on the one-sided Wilcoxon rank sum test (assuming a minimization problem). The probability of making a type I error is given by the significance level α, which, if not stated otherwise, is set to $\alpha = 0.01$.

When considering the results of multiple statistical hypothesis tests, it is very probable that some type I errors occur due to chance. If, for instance, 200 independent Wilcoxon rank sum tests are carried out, 2 type I errors are expected on average if all null hypotheses are true. The probability that there is at least one type I error computes to $1-(1-\alpha)^{200} = 1-0.99^{200} \approx 0.866$ (assuming true null hypotheses). This fact has to be kept in mind when interpreting the results of multiple statistical tests. Sometimes, correction methods such as the sequential Bonferroni procedure as proposed by Holm [Hol79] are used to cope with this situation. However, there are severe mathematical, logical and practical arguments against such methods [Mor03], and they are therefore not used in this thesis. Instead, the statistical results are interpreted with care and with this issue in mind.

For the computation of the Wilcoxon rank sum test, the function `wilcox.exact` (library `exactRankTests`, one-sided alternative `"less"`) of R [R D08] was used. Subsequently, the results of multiple Wilcoxon rank sum tests are often presented in tables. An example is given in Table 4.2.

Table 4.2: Example results of multiple one-sided Wilcoxon rank sum tests. For each algorithmic combination (A, B), the matrix shows on which functions algorithm A performed significantly better than algorithm B. The total number of benchmarks is 18.
Example: Entry {f5, f6} in the first row shows that Uniform significantly outperformed Zero (2) on benchmarks f5 and f6.

	1	2	3
Uniform (1)	{}	{f5, f6}	{}
Zero (2)	{Ackley, f2, f11, f12, f14}	{}	{}
Half-diff (3)	{Ackley, Rastrigin, f2, f3, f12, f14}	{Ackley, Rastrigin, f3}	{}

4.2 Test Functions

For the experimental evaluation, *traditional benchmarks* from the continuous optimization literature (e.g., [BK07, Cle06b, Men04, ES00]) and *CEC 2005 benchmarks* [SHL$^+$05] were used.

4.2.1 Traditional Benchmarks

The benchmark functions presented in this section are widely used in the PSO community, e.g., [BK07, Cle06b, Men04, ES00]. However, they have some drawbacks. First, the global optimum is often located at or very close to the center of the feasible space. Second, many of the problems are *separable*, i.e., their parameters do not influence each other. Algorithms can easily exploit separability by optimizing each functional parameter separately. When solely using benchmark functions that have their global optimum at or near the search space center, individuals are often initialized in a subspace of the feasible region in order to avoid that the performance of algorithms with center bias is overestimated [BK07]. E.g., the initialization space of the Sphere function may be set to $[50\ldots100]^n$ whereas the parameter space is $[-100\ldots100]^n$, where n denotes the search space dimensionality. As the CEC 2005 benchmarks contain shifted and rotated versions of the traditional benchmarks, asymmetric initialization ranges are not used in the subsequent experimental study. That way, the variability of the investigated problems is increased: Most problems do not have their global optimum at the center of the search space, but exceptions exist. Hence, center bias can easily be identified.

A function $f(x_1, x_2, \ldots x_n)$ is called *unimodal* if and only if it has a single local optimum which then, of course, also is the global optimum. Otherwise, it is called *multimodal*.

Sphere

Sphere is a very simple, separable, unimodal problem. The global minimum is $f(0,\ldots,0) = 0$.

$$\text{Function description:} \quad f(\vec{x}) = \sum_{i=1}^{n} x_i^2$$

$$\text{Search space:} \quad \mathcal{S} = [-100, 100]^n$$

Rosenbrock

The global optimum of the Rosenbrock function, $f(1,\ldots,1) = 0$, is located in a long, narrow, banana shaped valey, as depicted in Figure 4.2 (bottom, right). Rosenbrock is a non-separable problem.

$$\text{Function description:} \quad f(\vec{x}) = \sum_{i=1}^{n-1} \left(100 \cdot \left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2\right)$$

$$\text{Search space:} \quad \mathcal{S} = [-30, 30]^n$$

Ackley

The Ackley function has many local optima, and a deep value near the center of the feasible space. It is rather flat elsewhere. The global optimum is located at the origin: $f(0,\ldots,0) = 0$.

$$\text{Function description:} \quad f(\vec{x}) = -20 \exp\left(-0.2\sqrt{\tfrac{1}{n}\textstyle\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\tfrac{1}{n}\textstyle\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$$

$$\text{Search space:} \quad \mathcal{S} = [-32, 32]^n$$

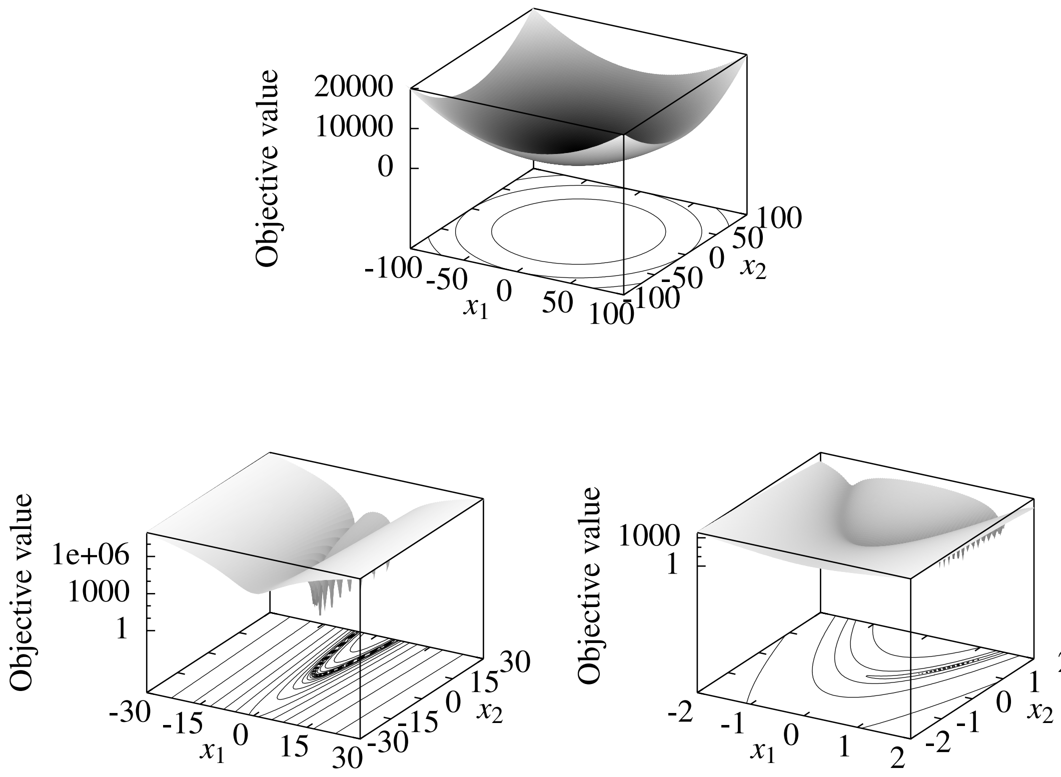


Figure 4.2: Top: The Sphere function. Bottom: The Rosenbrock function.

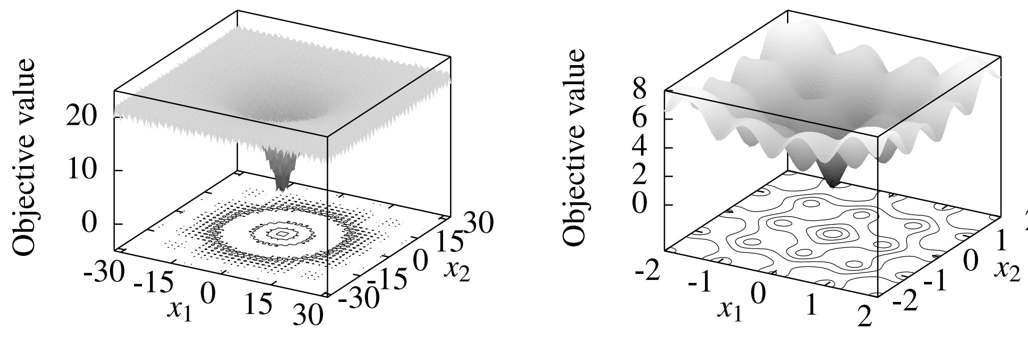


Figure 4.3: The Ackley function.

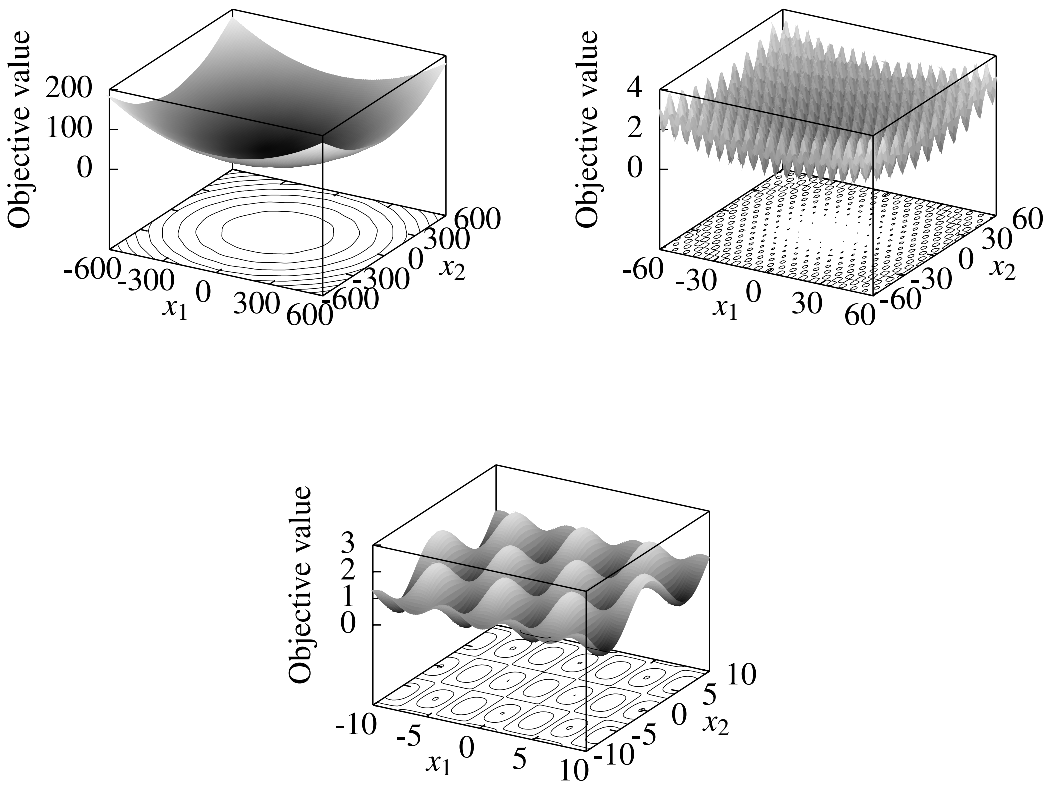


Figure 4.4: The Griewank function.

Griewank

Griewank is a multimodal problem with many regularly distributed local minima. From far, Griewank resembles the Sphere function, as shown in Figure 4.4. The global optimum is $f(0,\dots,0) = 0$.

$$\text{Function description:} \quad f(\vec{x}) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

$$\text{Search space:} \quad \mathcal{S} = [-600, 600]^n$$

Rastrigin

Rastrigin is a highly multimodal problem. There exist many local optima, which are regularly distributed over the feasible parameter space. Due to the high number of local optima, Rastrigin is suited to test the exploration capabilities of optimization algorithms. It is a separable problem. The global minimum is $f(0,\dots,0)=0$.

Function description: $f(\vec{x}) = 10 \cdot n + \sum_{i=1}^{n} \left(x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i)\right)$

Search space: $\mathcal{S} = [-5.12, 5.12]^n$

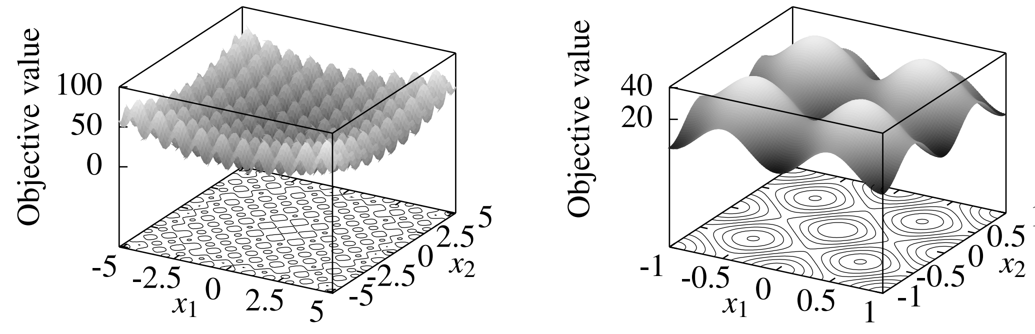


Figure 4.5: The Rastrigin function.

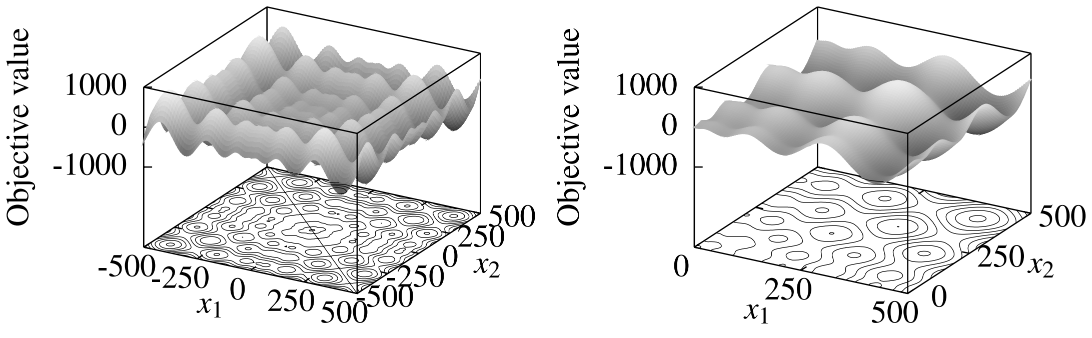


Figure 4.6: The Schwefel function.

Schwefel 2.6

Schwefel is the only problem of the traditional benchmarks whose global optimum is not located at or near the center of the feasible space. Instead, the global minimum is located at $x^* = (420.9687,\dots,420.9687), f(x^*) = -n \cdot 418.9829$. Schwefel is a separable multimodal problem with many local optima.

$$\text{Function description:} \quad f(\vec{x}) = \sum_{i=1}^{n} \left(-x_i \cdot \sin\left(\sqrt{|x_i|}\right)\right)$$
$$\text{Search space:} \quad \mathcal{S} = [-500, 500]^n$$

4.2.2 CEC 2005 Benchmarks

In 2005, the so-called *CEC 2005 benchmarks* were published [SHL$^+$05], which cover many different kinds of optimization problems. The CEC 2005 benchmarks are based on known functions like Sphere, Rosenbrock, Griewank, and Rastrigin. However, the functions were shifted by randomly generated offsets. This means that the global optima are distributed uniformly at random in the search space. It can be proved that, with overwhelming probability with respect to the search space dimensionality, they are located very close to the boundary:

Theorem 4.1. *Let f be a benchmark problem and $x^\star$ be a global optimum chosen uniformly at random in the n-dimensional search space bounded by $[-r,r]^n$. Then, the probability $p(r,n,\varepsilon)$ that the distance of $x^\star$ to its nearest border is less than ε is $1-e^{-\Theta(n)}$.*

Proof. Similarly to Theorem 3.1, $p(r,n,\varepsilon)$ evaluates to

$$p(r,n,\varepsilon) = \frac{(2r)^n - (2r-2\varepsilon)^n}{(2r)^n} = 1 - e^{-\Theta(n)} \qquad \square$$

The CEC 2005 benchmarks were designed to provide problems with various different features:

- Uni- and multimodality
- Separable and non-separable functions
- Benchmarks with and without noise
- Continuous and non-continous functions
- Benchmarks with and without search space boundaries

Additionally, each problem is scalable with respect to the search space dimensionality.

In the experimental analysis, the CEC 2005 benchmarks f1–f14 are considered. Functions without search space boundaries and with noise were excluded. The remaining problems are summarized in Table 4.3. More details, including function descriptions, can be found in the technical report of Suganthan et al. [SHL$^+$05], in which the CEC 2005 benchmarks were first defined.

Table 4.3: Summary of CEC 2005 benchmarks f1–f14 [SHL+05]. Each parameter is bounded by the values given in the column named *Range*, and $f(x^*)$ denotes the best objective value. The functions f4 and f7 were excluded because they either do not have box constraints or are noisy.

Unimodal functions	
f1	Shifted Sphere Function
f2	Shifted Schwefel's Problem 1.2
f3	Shifted Rotated High Conditioned Elliptic Function
f5	Schwefel's Problem 2.6 with Global Optimum on Bounds
Multimodal functions	
f6	Shifted Rosenbrock's Function
f8	Shifted Rotated Ackley's Function with Global Optimum on Bounds
f9	Shifted Rastrigin's Function
f10	Shifted Rotated Rastrigin's Function
f11	Shifted Rotated Weierstrass Function
f12	Schwefel's Problem 2.13
Expanded multimodal functions	
f13	Expanded Extended Griewank's plus Rosenbrock's Function (F8F2)
f14	Shifted Rotated Expanded Shaffer's F6

	$f(x^*)$	**Range**	**Features**
Unimodal functions			
f1	-450	$[-100,100]$	Unimodal, separable
f2	-450	$[-100,100]$	Unimodal, non-separable
f3	-450	$[-100,100]$	Unimodal, non-separable
f5	-310	$[-100,100]$	Unimodal, non-separable, global optimum on boundary
Multimodal functions			
f6	390	$[-100,100]$	Multimodal, non-separable
f8	-140	$[-32,32]$	Multimodal, non-separable, global optimum on boundary, needle-in-haystack character
f9	-330	$[-5,5]$	Multimodal, separable, many local optima
f10	-330	$[-5,5]$	Multimodal, non-separable, many local optima
f11	90	$[-0.5,0.5]$	Multimodal, non-separable
f12	-460	$[-\pi,\pi]$	Multimodal, non-separable
Expanded multimodal functions			
f13	-130	$[-5,5]$	Multimodal, non-separable
f14	-300	$[-100,100]$	Multimodal, non-separable

4.3 Velocity Initialization

In the theoretical analysis, three different velocity initialization strategies were investigated: uniform, zero, and half-diff. It was proved that none of the strategies prevents that many particles become infeasible in the first iteration, with overwhelming probability. If initial particle velocities are distributed uniformly at random in the underlying search space, almost certainly all particles leave the high-dimensional space. On the other hand, if velocities are initialized to zero, each particle i with $\vec{x}_{i,0} = \vec{p}_{i,0} = \vec{l}_{i,0}$ is unable to move until one of its neighbors improves. It seems that half-diff velocity initialization has the fewest drawbacks. Nevertheless, based on the theoretical analysis, similar results are expected for all three initialization strategies.

The effects of velocity initialization on particle swarm performance were investigated experimentally, by using 100-dimensional benchmarks, two different bound handling mechanisms, and the parameter set given in Table 4.1. As bound handling methods, Nearest-Z and Random-Z were chosen due to their complementary features: While particles with Nearest position handling preferably explore boundary regions, Random position handling often distracts particles from the boundary (see [HW07] and Section 4.4.2). Sample mean, respective 95% confidence intervals, and sample standard deviations s of the obtained final objective values from $N = 100$ runs per benchmark are given in Tables B.2 and B.3 in the Appendix. Note that the standard errors $s_{\overline{X}}$ can easily be computed to $s_{\overline{X}} = s/\sqrt{N} = s/10$.

Sample means and confidence intervals are additionally depicted in Figures 4.7 and 4.8. These plots show that most of the time, the sample means of the different velocity initialization strategies are similar. However, there are some notable exceptions, e.g., half-diff initialization achieves much better results on Rastrigin than the other two initialization procedures with Nearest-Z bound handling (see Figure 4.7). The statistical significance of visible differences in the obtained sample means was assessed by conducting one-sided Wilcoxon rank sum tests, as described in Section 4.1.3. In order to reduce the probability of type II errors (the probability that the null hypothesis is not rejected although it is false) the significance level was increased to $\alpha = 0.05$. The summarized results are shown in Tables 4.4 and 4.5.

When Random-Z bound handling is utilized, none of the velocity initialization strategies performed particularly good or bad compared to the other strategies, considering the results of the Wilcoxon rank sum tests (see Tables 4.4 and 4.5). On the investigated testproblems, the impact of velocity initialization is increased when using Nearest-Z bound handling: Significant differences were observed on a total of 9 benchmark functions. In this setting, half-diff velocity initialization produced slightly better results than the other two methods on many benchmarks. Possible reasons were discussed above. Due to this argumentation, from the investigated methods half-diff velocity initialization can be recommended for practical PSO application, although usually, large performance differences are not expected.

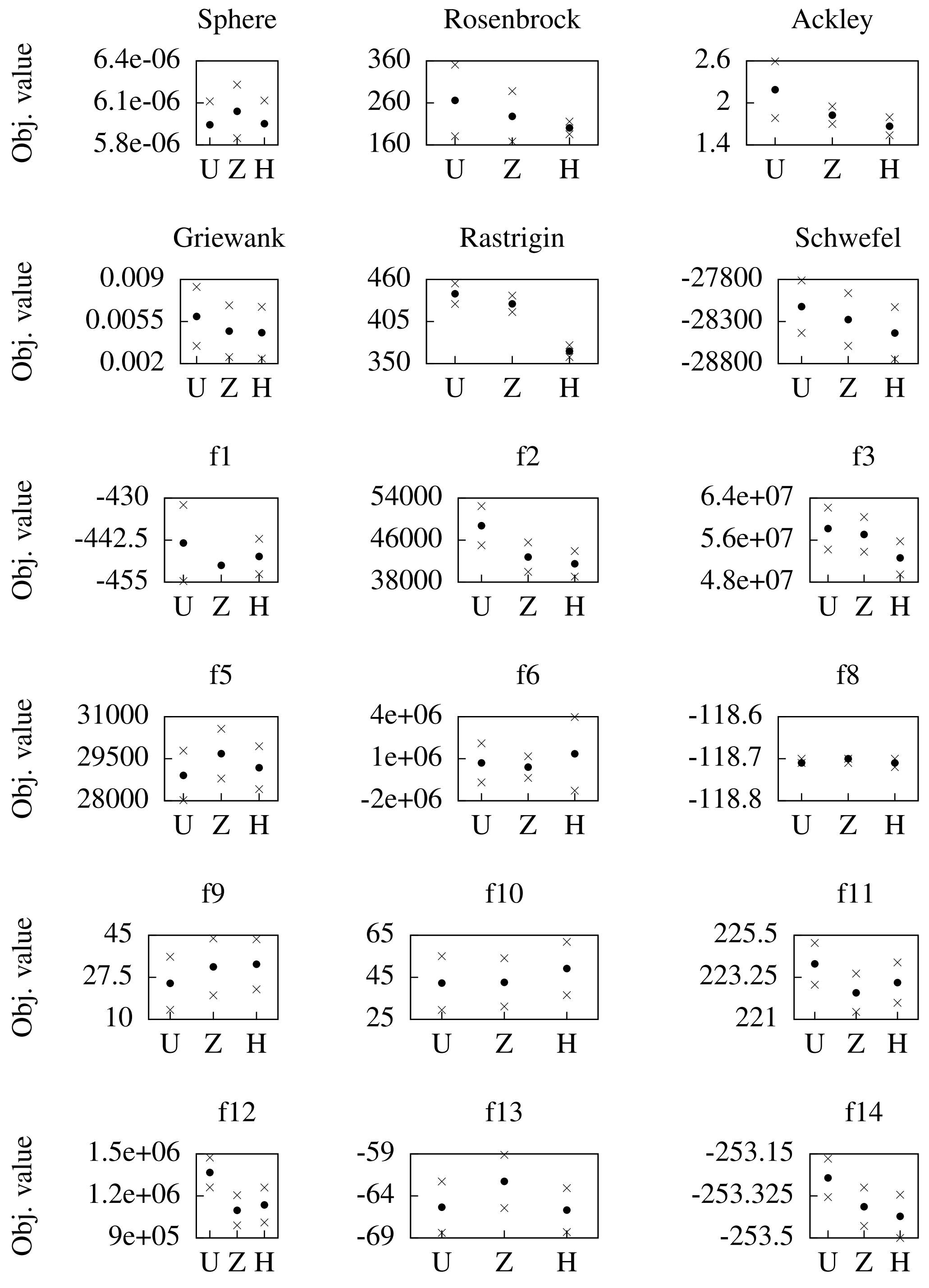


Figure 4.7: Comparison of velocity initialization strategies: Uniform (U), Zero (Z), and Half-diff (H). Sample mean of final objective values (dot) and corresponding 95% confidence intervals (X) are shown for each 100-dimensional benchmark. **Nearest-Z** bound handling was used.

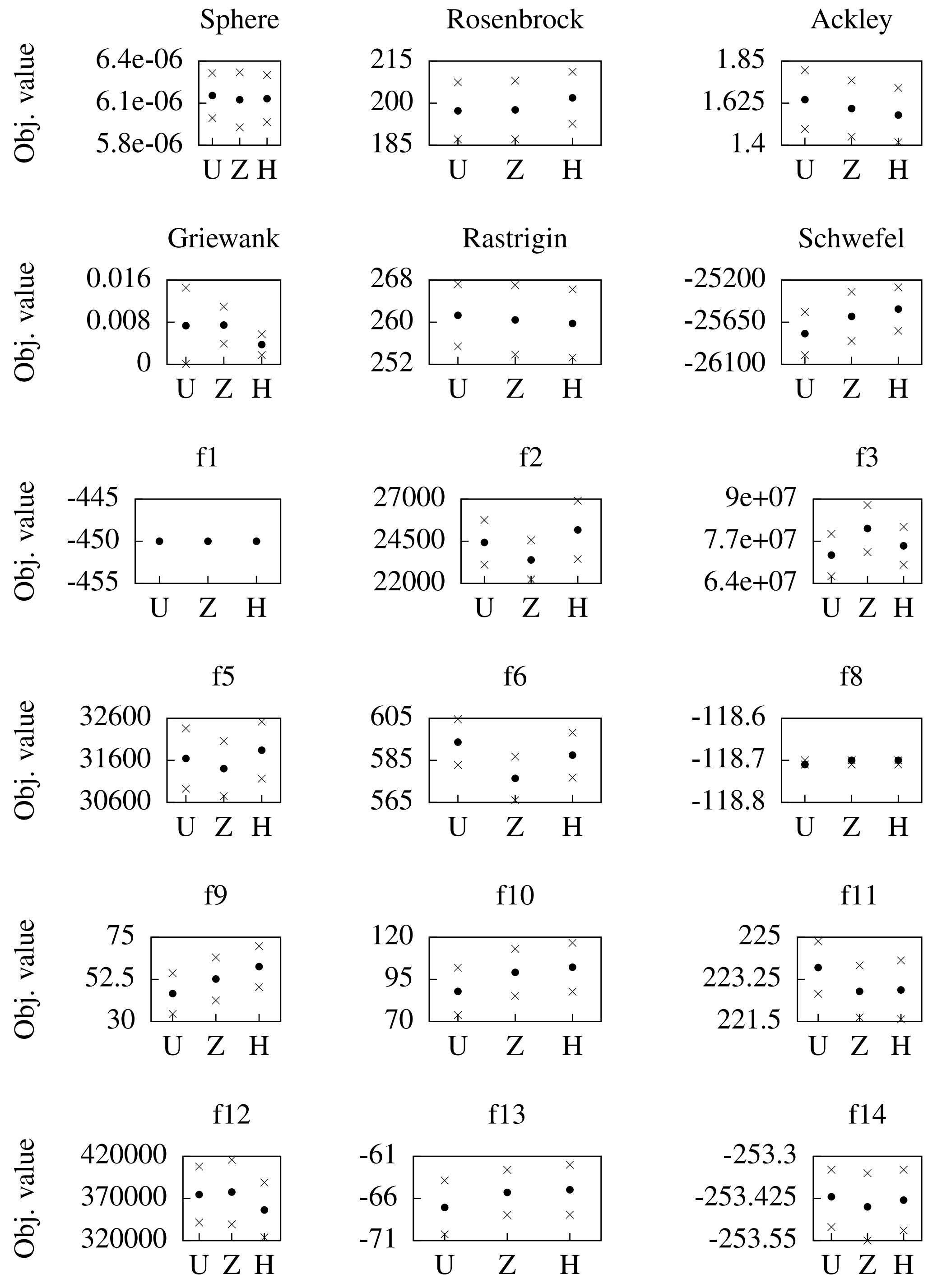


Figure 4.8: Comparison of velocity initialization strategies: Uniform (U), Zero (Z), and Half-diff (H). Sample mean of final objective values (dot) and corresponding 95% confidence intervals (X) are shown for each 100-dimensional benchmark. **Random-Z** bound handling was used.

Table 4.4: Summary of one-sided Wilcoxon rank sum test with significance level $\alpha = 0.05$. For each algorithmic combination (A, B), this matrix shows on which functions A performed significantly better than B. The total number of benchmarks is 18.

Nearest-Z

	1	2	3
Uniform (1)	{}	{f5, f6}	{}
Zero (2)	{Ackley, f2, f11, f12, f14}	{}	{}
Half-diff (3)	{Ackley, Rastrigin, f2, f3, f12, f14}	{Ackley, Rastrigin, f3}	{}

Random-Z

	1	2	3
Uniform (1)	{}	{Griewank}	{f9}
Zero (2)	{f6}	{}	{f6}
Half-diff (3)	{}	{Griewank}	{}

Table 4.5: Summary of one-sided Wilcoxon rank sum test with significance level $\alpha = 0.05$. For each algorithmic combination (A, B), this matrix shows how often algorithm A performed significantly better than algorithm B. The total number of benchmarks is 18.

Nearest-Z

	1	2	3
Uniform (1)	0	2	0
Zero (2)	5	0	0
Half-diff (3)	6	3	0

Random-Z

	1	2	3
Uniform (1)	0	1	1
Zero (2)	1	0	1
Half-diff (3)	0	1	0

4.4 Bound Handling

The experimental analysis presented in this section has two main purposes: First, the significance of bound handling for particle swarm optimization is analyzed, in order to back up the theoretical results obtained in Section 3. Second, a more practical issue is addressed: The strengths and weaknesses of several commonly-used bound handling strategies are analyzed.

The first topic is discussed in Section 4.4.1 by studying five different bound handling strategies on the chosen benchmark set. To investigate the impact of problem dimensionality on the significance of bound handling, the parameter space dimensionality is set to 2, 30, 100, and 500. Clearly, considering the theoretical results, we expect that the significance of bound handling increases with the problem dimensionality. First experimental results on the importance of bound handling were published previously [HW08]. In the following, these results are extended by using additional bound handling methods and problem dimensionalities.

The characteristics of 13 commonly-used bound handling strategies are analyzed in Sections 4.4.2 and 4.4.3. Guidelines for PSO application on high-dimensional problems with box constraints are derived.

Throughout this section, the algorithmic parameters were chosen according to Table 4.1. Particle velocities were initialized with half-diff initialization due to the fact that it provided slightly better results than zero and uniform in the experiments presented in Section 4.3.

4.4.1 Significance of Bound Handling

The theoretical study showed that many particles become infeasible at the beginning of the optimization process. As bound handling is applied to each of these particles, the chosen strategy to cope with infeasible solutions has strong impact on particle swarm behavior, at least in the early steps of a PSO algorithm. In this section, the significance of bound handling for particle swarm optimization is investigated experimentally.

Five bound handling strategies (RandomBack, Nearest-Z, Random-Z, Reflect-Z, and Infinity) were studied on the benchmarks that were presented in the previous section. Descriptions of the applied bound handling methods can be found on page 87 and in Section 2.3.2. Velocity handling is analyzed seperately in Section 4.4.3. Therefore, Nearest-A, Nearest-U, Random-A, Random-U, Reflect-A, and Reflect-U were excluded from this experiment. Hyperbolic and Infinity-C are not bound handling methods in the strong sense because they are not only applied to infeasible particles but to all particles. As this fact could already lead to significant performance differences, Hyperbolic and Infinity-C were excluded from the experiment as well.

Tables of sample means, 95% confidence intervals, and standard deviations can be found in Appendix B.3. The tables include Hyperbolic and Infinity-C because these results are needed in the subsequent section. However, the performance of Hyperbolic and Infinity-C is disregarded for the conclusions drawn in this section.

A comparison of the obtained sample means shows that bound handling strongly influenced particle swarm performance on the higher-dimensional problems. Selected results are presented in Table 4.6 for convenience. The obtained average objective values hardly differ from one another for two-dimensional problems. This is partly due to the fact that the two-dimensional problems are comparatively easy to solve, and the PSO algorithm was mostly able to find the global minimum. When solving 30-dimensional problems, the performance differences are more noticeable (Tables 4.6, B.14, and B.15) and significant (Tables 4.7 and B.7). The impact of bound handling on the final solution quality increased further with the problem dimensionality. The observed performance differences are significant from a statistical point of view: Table 4.7 shows the summarized results of the one-sided Wilcoxon rank sum test for 30- and 500-dimensional problems, using Random-Z, Reflect-Z, and Infinity bound handling. The complete results of the Wilcoxon rank sum tests are presented in Appendix B.3. A large number of significant performance differences was observed, especially when considering the higher-dimensional benchmark sets.

Figure 4.9 shows how often a bound handling strategy significantly outperformed another strategy, for each dimensionality. Two main conclusions can be drawn: First, this number increases with the problem dimensionality, as suggested by the theoretical analysis. Second, bound handling significantly influenced particle swarm performance in this experimental investigation. For instance, a pairwise comparison of the studied bound handling strategies led to 127 observed significant performance differences for the 100-dimensional benchmarks, while the greatest possible number of significant results is 180.

Summarized, this experiment shows that bound handling not only influences particle swarm behavior in the early steps of a PSO algorithm, as proved in the theoretical study, but also has strong impact on the final solution quality of a particle swarm optimizer, in particular, when solving high-dimensional problems. Hence, the choice of a suitable bound handling mechanism is important for practical PSO application. In the subsequent section, the characteristics of seven commonly-used bound handling methods are analyzed in detail, to assist in the process of selecting an appropriate bound handling strategy for a given problem.

4.4.2 Strengths and Weaknesses of Selected Strategies

In this section, the characteristics of seven commonly-used bound handling strategies are analyzed in detail, by using both traditional and CEC 2005 benchmarks. The parameter space dimensionality was varied and set to 2, 30, 100, and 500 dimensions for each problem. The main goal of this investigation is to provide guidelines for the selection of a suitable bound handling strategy for a given problem. Sometimes, a pri-

Table 4.6: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies for selected benchmarks (full results can be found in Appendix B.3). The best objective values are presented together with the function name.

2D	Rastrigin (0)	f1 (-450)	f10 (-330)
Hyperbolic	4.3e-12±7.1e-13 (3.6e-12)	**-450 (0)**	**-330 (0)**
RandomBack	*4.4e-12±9.4e-13 (4.7e-12)*	**-450 (0)**	**-330 (0)**
Nearest-Z	**3.6e-12±8.2e-13 (4.1e-12)**	**-450 (0)**	**-330 (0)**
Random-Z	4.2e-12±8.0e-13 (4.0e-12)	**-450 (0)**	**-330 (0)**
Reflect-Z	3.8e-12±7.1e-13 (3.6e-12)	**-450 (0)**	**-330 (0)**
Infinity	4.2e-12±9.1e-13 (4.6e-12)	**-450 (0)**	**-330 (0)**
Infinity-C	3.6e-12±7.4e-13 (3.7e-12)	**-450 (0)**	**-330 (0)**
30D	Rastrigin (0)	f1 (-450)	f10 (-330)
Hyperbolic	**28.874±1.4589 (7.3523)**	**-450 (0)**	*-206.7±4.493 (22.64)*
RandomBack	42.684±1.9667 (9.9118)	**-450 (0)**	-263.8±3.17 (15.98)
Nearest-Z	51.549±2.7688 (13.954)	**-450 (0)**	-264.9±3.265 (16.45)
Random-Z	40.783±1.9356 (9.7552)	**-450 (0)**	-266.5±3.200 (16.13)
Reflect-Z	*52.474±2.7422 (13.82)*	**-450 (0)**	**-274.6±2.582 (13.02)**
Infinity	49.529±2.4716 (12.456)	**-450 (0)**	-228.17±5.41 (27.24)
Infinity-C	38.973±2.0172 (10.166)	**-450 (0)**	-254.76±3.63 (18.31)
100D	Rastrigin (0)	f1 (-450)	f10 (-330)
Hyperbolic	**105.46±4.1927 (21.13)**	**-450 (0)**	478.7±15.56 (78.4)
RandomBack	337.09±8.0286 (40.462)	**-450 (0)**	104.2±13.0 (65.51)
Nearest-Z	366.22±7.7675 (39.146)	-447.36±5.2478 (26.447)	49.19±12.66 (63.79)
Random-Z	259.75±6.483 (32.673)	**-450 (0)**	102.2±14.47 (72.9)
Reflect-Z	355.61±7.7081 (38.847)	**-450 (0)**	**8.256±12.61 (63.54)**
Infinity	*745.71±58.688 (295.77)*	*188740±20674 (104190)*	*1905±73.08 (368.3)*
Infinity-C	262.51±6.5867 (33.195)	**-450 (0)**	285.3±21.16 (106.7)
500D	Rastrigin (0)	f1 (-450)	f10 (-330)
Hyperbolic	**467.81±12.732 (64.167)**	**-417.69 ±2.2298 (11.238)**	7320±61.41 (309.5)
RandomBack	2360.3±40.692 (205.08)	3350.5±481.59 (2427.1)	5940.5±122.5 (617.4)
Nearest-Z	2435.5±33.708 (169.88)	7009.6±938.12 (4727.9)	5043.5±99.93 (503.6)
Random-Z	1804.2±34.68 (174.78)	23251±1449.8 (7306.5)	5684.7±103.1 (519.3)
Reflect-Z	2341.1±31.406 (158.28)	2269.2±349.6 (1761.9)	**4460.7±112.8 (568.3)**
Infinity	*6727.1±20.394 (102.78)*	*2184100±11527 (58096)*	*16910±69.90 (352.3)*
Infinity-C	6725.7±20.941 (105.54)	2174800±13203 (66538)	16871±74.97 (377.8)

Table 4.7: Summary of one-sided Wilcoxon rank sum test with significance level 0.01 for the bound handling strategies Random-Z, Reflect-Z, and Infinity. For each algorithmic combination (A, B), this matrix shows on which benchmarks A performed significantly better than B. $\mathcal{B}$ denotes the set of all 18 benchmarks.

30D	Random-Z	Reflect-Z	Infinity
Random-Z	{}	{Ra, f12}	{Ra, Schw, f5, f9, f10, f11, f12, f13, f14}
Reflect-Z	{Schw, f5, f10}	{}	{Schw, f5, f9, f10, f11, f13, f14}
Infinity	{f3}	{f3}	{}
500D	Random-Z	Reflect-Z	Infinity
Random-Z	{}	{Sphere, Rosenbrock, Ackley, Griewank, Rastrigin, f5, f13}	$\mathcal{B}$
Reflect-Z	{Schwefel, f1, f2, f3, f6, f10, f11, f12}	{}	$\mathcal{B}$
Infinity	{}	{}	{}

ori knowledge is available when solving real world problems, which can be exploited in the parameter selection process. If a priori knowledge is not available, adaptive particle swarm optimizers can be used. Two adaptive approaches are presented in Chapter 5.

A Modified Velocity Update Operator: Hyperbolic

In Section 2.3.2, the existing bound handling strategies were classified into repair methods, special problem representations, and special velocity update operators. Hyperbolic is a special velocity update operator, which, by adjusting the particles' velocities according to the given box constraints, prevents that particles become infeasible.

The sample means of the obtained final objective values and the results of the Wilcoxon rank sum test (see Appendix B.3, selected results are presented in Table 4.6 for convenience) show that Hyperbolic often clearly outperformed the other bound handling strategies when solving high-dimensional problems. There are only three functions on which Hyperbolic was repeatedly outperformed by other methods: f9, f10, and Schwefel. The situation is different when solving two-dimensional problems: Hyperbolic was significantly outperformed by the other strategies on Schwefel and f8. These two functions have their global optimum located at or near the search space boundary. Note that only few significant results were observed in the two-dimensional case (see Table B.4). Hence, the weak performance on some two-dimensional problems is a clear drawback of Hyperbolic.

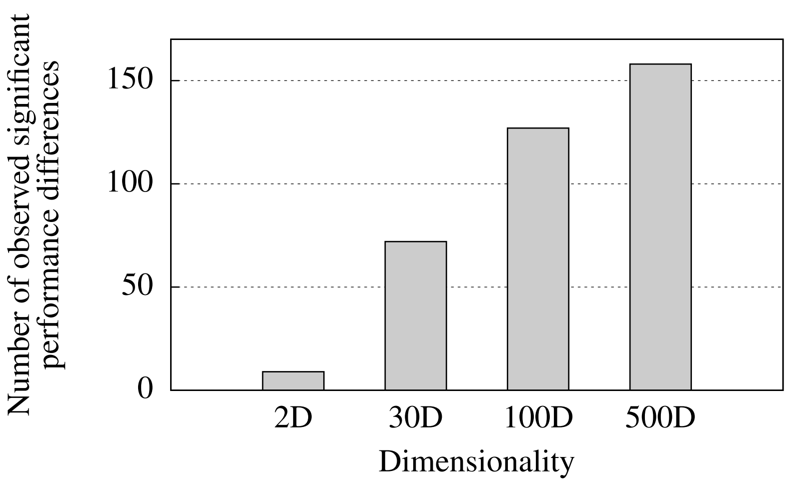


Figure 4.9: The higher the problem dimensionality, the more frequently significant performance differences were observed among the investigated bound handling strategies. The greatest possible number of significant observations is 180.

A possible reason for the good performance of Hyperbolic on the higher-dimensional benchmarks can directly be derived from the theoretical analysis. The results presented in Section 3.4 indicate that small particle velocities are advantageous when solving high-dimensional problems. The Euclidean distance of each particle's movement was logged per iteration, and averaged at the end of an optimization run. Sample means and corresponding standard errors are shown in Table 4.8 for a representative set of 100-dimensional benchmarks. With Hyperbolic bound handling, particle velocities were relatively small. Small velocities lead to diminished exploration, but enhanced exploitation capabilities.

Although Hyperbolic provided outstandingly good solutions for most high-dimensional problems, there are exceptions: Schwefel, f9, and f10. The CEC benchmarks f9 and f10 have a huge number of local optima [SHL$^+$05]. The swarm might be attracted to one of these local optima too early due to the small particle velocities.

The experimental study suggests that, from the investigated methods, Hyperbolic can be recommended for high-dimensional problems with a moderate number of local optima. Hyperbolic can be disadvantageous for low-dimensional problems, and it seems that Hyperbolic is unable to cope with a large number of local optima.

Bound Resetting Methods: RandomBack and Nearest-Z

Nearest position handling might lead to premature convergence on the boundary, as pointed out previously [ZXB04, HW07] (see also Section 3.5). This effect can be reduced by choosing the algorithmic PSO parameters such that exploration is in-

Table 4.8: With Hyperbolic bound handling, particles move considerably slower through the search space. In this table, the average distance a particle moved throughout the run and corresponding 95% confidence interval are shown for some 100-dimensional problems.

	f1	f6	f10	f14
Hyperbolic	3.467±0.009267	4.284±0.02203	1.704±0.02669	59.94±0.5966
RandomBack	14.19±0.04910	21.45±0.1021	7.152±0.07178	318.0±1.998
Nearest-Z	11.33±0.03892	17.15±0.1333	5.552±0.05691	242.5±1.431
Random-Z	15.01±0.05398	20.23±0.1032	5.702±0.05052	211.0±1.117
Reflect-Z	9.598±0.03077	14.2±0.0537	4.944±0.04254	201.2±1.182
Infinity	723.2±4.109	714.3±4.702	36.04±0.2082	801.9±3.766
Infinity-C	18.19±0.1513	22.31±0.1364	5.294±0.06076	191.6±1.277

creased, for instance, by using a sparsely connected neighborhood graph. Moreover, the inversion of an infeasible particle's velocity, as it is done in RandomBack, can help to prevent this kind of premature convergence. Nevertheless, the search is biased towards the boundary. This holds in particular if the underlying parameter space is of high dimensionality due to the fact that the probability of infeasible particles increases in that case (at least at the beginning of the optimization, as shown in the theoretical study in Section 3.3). This means that boundary solutions are evaluated comparatively often, which biases the search in that direction. Note that there are scenarios in which this swarm behavior can pay off, e.g., if good solutions are located on the boundary. However, generally, the preference of boundary regions is not a desired feature.

Both RandomBack and Nearest-Z only showed mediocre performance in the experimental investigation, considering both sample means and the results of the Wilcoxon rank sum test (see Appendix B.3). There are only very few benchmarks on which Nearest-Z or RandomBack significantly outperformed Reflect-Z (500-dimensional Ackley, 30- and 100-dimensional Rastrigin), which suggests that Reflect-Z might be the better choice for many scenarios.

Random-Z

Random position handling might distract particles from the boundary, and lead to poor performance if good solutions are located at or near the search space border, as observed earlier [HW07]. The reason is clear: Whenever a particle approaches the boundary, but moves slightly to far, it is possibly replaced at a completely different position in the parameter space, which can lead to a very exploratory, rather random swarm behavior. The swarm's difficulty to approach the boundary can already be observed when solving two-dimensional problems: Random-Z was significantly outperformed by most other approaches on f5 and f8 (see results of the Wilcoxon rank sum test, Table B.4). Both functions have their global optimum at the boundary [SHL$^+$05]. Considering f5, Random-Z was the only strategy that had not found the global optimum in all runs.

The previous argumentation is confirmed by the results of the 500-dimensional benchmarks: Random-Z often significantly outperformed the other bound handling strategies, in particular on Sphere, Rosenbrock, Ackley, Griewank, and Rastrigin, which have their global optimum at or near the search space center. On the other hand, Random-Z is often significantly outperformed on Schwefel and some CEC 2005 benchmarks. These results indicate that the application of Random-Z biases the particle swarm towards the search space center.

Due to its exploration capabilities, Random-Z might be a good choice for highly multimodal problems. However, this thought was not confirmed by experimental results. The performance on f9 and f10 was not competitive with, for instance, Reflect-Z, considering the sample means obtained for the 30-, 100-, and 500-dimensional benchmarks (see Appendix B.3).

In Theorem 4.1 it was shown that in high-dimensional spaces, the global optimum is located very close to the boundary if its position is assumed to be distributed uniformly at random in the search space. Moreover, in high-dimensional spaces, most of the volume is concentrated in a small shell near the surface (see Theorem 3.1). This means that an optimization algorithm should be able to explore regions near the boundary. Therefore, Random-Z bound handling might be detrimental when solving high-dimensional problems. Moreover, if good solutions are located at or near the search space boundary, particles using Random-Z bound handling can have difficulties to approach them. However, Random-Z often achieved very good results on the traditional benchmarks that have their global optimum (and other good solutions) at or near the search space center.

Reflect-Z

Reflect-Z was among the best performing strategies in this experimental investigation. Its design does neither prevent particles from approaching the search space borders nor explicitly attract particles towards the boundary. It seems that Reflect position handling carries none of the disadvantages that were observed when applying Random or Nearest. Moreover, in contrast to the periodic search space of Zhang et al. [ZXB04] (see also Section 2.3.2) there are no new discontinuities at the boundary, and the search space volume is not increased.

Particle velocities are in the same order of magnitude as those of Random-Z or Nearest-Z bound handling (see Table 4.8). This is probably the reason why Hyperbolic often significantly outperformed Reflect-Z on many higher-dimensional problems. However, Reflect-Z performed best (once: second-best) on the 30-, 100-, and 500-dimensional Schwefel, f9, and f10 (the results are presented in Appendix B.3).

The performance of Reflect-Z might be improved if velocities are decreased. For instance, velocity adaptation as presented in Section 5.3 can be used.

Special Problem Representation: Infinity and Infinity-C

In this experimental study, Infinity could not compete with the other strategies on higher-dimensional problems. On the 100- and 500-dimensional benchmarks, Infinity was significantly outperformed by all other strategies on all investigated testfunctions. The obtained average objective values were very poor (see Tables 4.6 and 4.7, complete results can be found in Appendix B.3). Note that each run of the PSO algorithm was terminated after 300,000 function evaluations. The evaluation step was skipped for infeasible particles which means that the number of iterations can be higher with Infinity bound handling than with, for instance, a repair method.

In the theoretical study presented in Section 3.3, it was shown that many particles become infeasible in the first step of a PSO algorithm, when solving high-dimensional problems. This means that many particles mainly explore infeasible space at the beginning of the optimization if Infinity bound handling is applied. Although the evaluation step is skipped, this might have impact on the interactions that usually take place in a particle swarm, as infeasible particles do not update their private guides. Moreover, particle velocities were comparatively high with Infinity bound handling, as shown in Table 4.8.

On the 100-dimensional benchmarks, velocity clamping strongly improved particle swarm performance. Mostly, both the obtained average objective values and the average distances covered by a particle's movement were roughly in the same order of magnitude than those of the repair mechanisms. However, velocity clamping did not significantly improve particle swarm performance on the 500-dimensional problems (see results of the Wilcoxon rank sum test presented in Appendix B.3).

The performance of Infinity and Infinity-C was competitive with those of the other bound handling strategies on the two- and 30-dimensional benchmarks.

Conclusion

The experimental analysis presented in this section revealed some of the strengths and weaknesses of diverse commonly-used bound handling strategies. In some real world applications, a priori knowledge is available about the properties of the optimization problem. This knowledge can be exploited when selecting the algorithmic PSO parameters.

Summarized, Hyperbolic often performed very good on the high-dimensional problems, as illustrated in Figure 4.10, possibly due to the fact that particle velocities are very small. Nevertheless, as exemplarily shown in Figure 4.11, there exist exceptions, for instance f9 and f10, which have a large number of local optima. The bound resetting methods RandomBack and Nearest-Z showed mediocre performance in this experimental investigation. Random position handling can distract particles from the boundary. However, good results were achieved on most traditional benchmarks, which have their global optimum and other good solutions near the search space center. Reflect-Z showed good overall performance, but was not competitive with Hyperbolic on most high-dimensional problems. The performance of Reflect-Z can be improved by using the velocity adaptation mechanism presented in Section 5.3. Both Infinity, which was proposed as bound handling strategy for standard particle swarm optimization [BK07], and Infinity-C were significantly outperformed by all other strategies on all 100- and 500-dimensional benchmarks. The obtained average objective values were by far worse than the ones obtained by the other strategies (see also Figures 4.10 and 4.11).

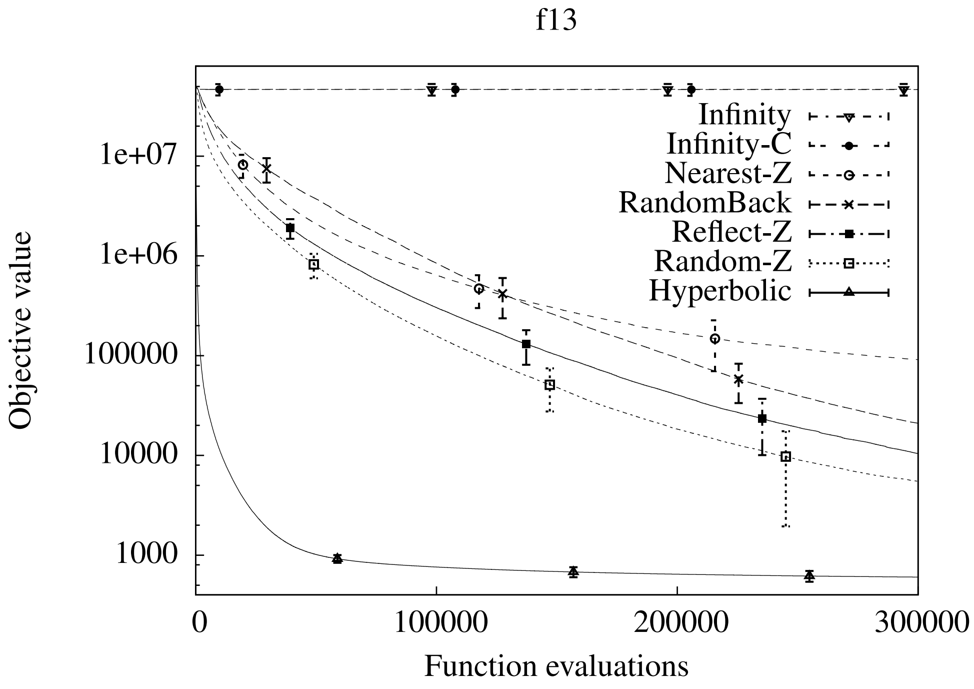


Figure 4.10: Convergence plot for the 500-dimensional f13 problem. Often, Hyperbolic led to considerably superior results than the other investigated methods. Infinity was significantly outperformed by all other methods on most high-dimensional (100 and 500 dimensional) problems. In the plot, sample means and standard deviations (vertical bars) are shown. Note that a logarithmic scale is used.

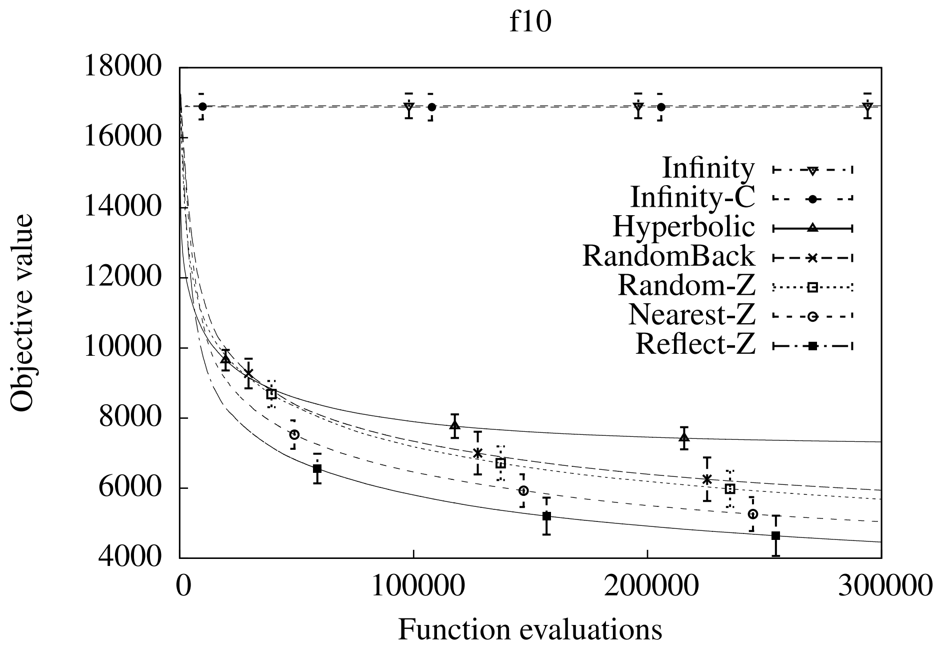


Figure 4.11: Convergence plot for the 500-dimensional f10 problem (shifted rotated Rastrigin). Hyperbolic significantly outperformed most bound handling strategies on many problems. However, on f9, f10, and Schwefel, Hyperbolic bound handling performed significantly worse than most other strategies.

4.4.3 Velocity Handling

In Section 2.3.2, several commonly-used bound handling methods were presented, among them repair algorithms. In particle swarm optimization, repair algorithms affect both a particle's position and its velocity. In the following experiment, the impact of velocity handling on particle swarm performance is examined, and the consequences for practical PSO application are discussed. The following strategies are considered, and tested on all 100-dimensional benchmarks[2]:

- Nearest-Z, Nearest-A, Nearest-U
- Random-Z, Random-A, Random-U
- Reflect-Z, Reflect-A, Reflect-U

The remaining parameters of the PSO algorithm are given in Table 4.1. Half-diff velocity initialization was used. Sample means, 95% confidence intervals, and

[2]Remember that *-Z*, *-A*, and *-U* are abbreviations for *Zero*, *Adjust*, and *Unmodified*. These velocity handling methods are described in Section 2.3.2 on page 30.

sample standard deviations of the obtained final objective values from $N = 100$ runs per benchmark are presented in Tables B.21, B.22, and B.23 in the Appendix. The results of the one-sided Wilcoxon rank sum tests with significance level $\alpha = 0.01$ are summarized in Tables B.20 and 4.9.

Table 4.9: Summary of one-sided Wilcoxon rank sum test with significance level $\alpha = 0.01$. The total number of benchmarks is 18.

	1	2	3
Nearest-Z (1)	0	8	11
Nearest-A (2)	0	0	10
Nearest-U (3)	0	0	0

	4	5	6
Random-Z (4)	0	5	12
Random-A (5)	2	0	12
Random-U (6)	1	1	0

	7	8	9
Reflect-Z (7)	0	1	12
Reflect-A (8)	0	0	12
Reflect-U (9)	1	1	0

Table 4.9 shows that Unmodified velocity handling cannot compete with the other two strategies on the tested benchmark set. The sample means obtained by Nearest-U, Random-U and Reflect-U were often considerably worse than those obtained with Zero and Adjust velocity handling. There are manifold reasons that explain these experimental results:

- Nearest position handling: If particle i is infeasible in iteration t, it violates the upper or lower search space boundary in at least a certain dimension d. Assume that the upper search space boundary is violated, then $x_{i,t,d} > ub_d$ holds before the particle is repaired. Furthermore, $lb_d \leq x_{i,t-1,d} \leq ub_d$ holds at the end of iteration $t-1$ as infeasible particles are repaired. As $x_{i,t,d}$ was computed from Equation (2.2) to $x_{i,t,d} = x_{i,t-1,d} + v_{i,t,d}$, $v_{i,t,d} > 0$ holds. As $x_{i,t,d} = ub_d$ after the application of the repair mechanism, the particle is again attracted towards infeasible space in the next iteration if $v_{i,t,d} > 0$ is not altered. This may lead to premature convergence on the boundary, in particular, if a densely connected neighborhood topology is used. The phenomenon of premature convergence on the boundary was already experimentally observed earlier [ZXB04], and investigated theoretically in Section 3.5.

  A similar argumentation explains why also Adjust velocity handling was disadvantageous in conjunction with Nearest position handling (see Tables B.20, B.21, B.22, and B.23).

  Alvarez-Benitez et al. [ABEF05] propose a bound handling strategy called *SHR*

(Shrink) which scales an infeasible particle's velocity vector such that the particle exactly arrives on the boundary. This method is similar to Nearest-A and was therefore not included in the experimentation. Note, however, that from the three investigated velocity handling methods, Zero should preferably be used in combination with Shrink due to the above argumentation, which holds for Shrink as well.

- Random and Reflect position handling: With Unmodified velocity handling, particle velocities are generally much larger than with Zero or Adjust velocity handling, as shown in Table 4.10. As a consequence, particles more often become infeasible. Table 4.11 shows how often bound handling was applied on average, for each strategy. This quantity equals the number of infeasible particles throughout an optimization run. Clearly, Unmodified velocity handling led to the highest number of infeasible particles for all three position handling strategies. Both the theoretical study and the experimental results have demonstrated that small velocities are advantageous for high-dimensional optimization problems, which explains the bad performance of Unmodified velocity handling on the given testfunctions.

The comparison of *Zero* and *Adjust* velocity handling leads to the following conclusions: In combination with Nearest position handling, *Zero* velocity handling is to be preferred due to the argumentation stated above and the experimental results. If Reflect position handling is utilized, *Zero* and *Adjust* velocity handling performed equally well on the tested benchmark set (see Tables 4.9 and B.23). When using Random position handling, *Zero* performed slightly better than *Adjust* (Tables 4.9 and B.22).

Summarized, in this experimental setup, *Unmodified* performed significantly worse than the other two methods, while *Zero* velocity handling constantly resulted in comparatively good solution quality. In combination with Reflect position handling, both *Zero* and *Adjust* velocity handling can be used.

4.5 Concluding Remarks

In this chapter, the results of the theoretical study were confirmed and complemented by thorough experimental analyses. Thirteen bound handling mechanisms were investigated on 18 benchmark functions. The experimental outcome was analyzed by statistical methods, such as hypothesis testing and the computation of confidence intervals.

As suggested by the theoretical analysis, velocity initialization had only slight impact on particle swarm performance, while bound handling significantly influenced the final solution quality provided by a particle swarm optimizer. This means that bound handling strategies have to be carefully selected, in accordance to the given

Table 4.10: Average distance a particle moved per iteration and corresponding 95% confidence intervals.

	f1	f6	f10	f14
Nearest-Z	11.332±0.07722	17.149±0.26444	5.552±0.11291	242.52±2.839
Nearest-A	11.269±0.1079	17.404±0.3119	5.188±0.093979	224.91±2.952
Nearest-U	11.715±0.1416	18.236±0.32019	5.000±0.08303	218.82±2.861
Random-Z	15.014±0.1071	20.225±0.20486	5.702±0.10024	210.99±2.217
Random-A	21.287±0.2541	26.735±0.3641	5.988±0.097678	224.31±2.363
Random-U	244.24±1.480	225.57±1.3066	9.618±0.15988	323.04±2.674
Reflect-Z	9.5978±0.06105	14.198±0.1065	4.9439±0.084414	201.18±2.345
Reflect-A	9.5088±0.0566	14.227±0.1345	4.9256±0.093283	198.34±2.227
Reflect-U	175.59±2.419	177.32±2.037	8.7806±0.15187	272.02±2.507

Table 4.11: Average number of infeasible particles for some 100-dimensional benchmarks and 95% corresponding confidence intervals.

	Rastrigin	f1	f10	f13
Nearest-Z	97287±4148.6	10003±134.5	128630±2369.1	17683±1464.8
Nearest-A	95815±4174.6	14395±901.36	141400±2150.9	16855±1225.7
Nearest-U	102170±4289.4	41647±2868.4	164480±2372.1	18234±1112.5
Random-Z	84221±3821.3	16608±192.2	118800±3026.6	12731±1037.4
Random-A	87382±3275.6	28275±486.63	126370±3087.9	13256±961.52
Random-U	104370±3383.7	299370±198.8	254730±4092.3	29184±1748.8
Reflect-Z	90476±3866.2	7794.5±85.46	104220±1773	12507±951.82
Reflect-A	94022±4128	8645.3±105.71	116130±2290.1	11839±766.67
Reflect-U	100530±3635.1	285850.0±1681.6	253540±2944	20626±1588.5

optimization problem. In order to assist in this process, the characteristics of several commonly-used bound handling strategies were analyzed in detail. Some guidelines for practical PSO application were derived. If the properties of the optimization problem are not available, the algorithmic parameters can be automatically adapted during the optimization process by using the adaptive Multi-Swarm PSO with Migration, which is presented in the next chapter.

5. Adaptive Particle Swarm Optimization

Most theoretical analyses of particle swarm optimization consider the convergence, stability, or runtime behavior of a particle swarm (see Section 2.4). Convergence analyses [CK02, Tre03, JLY07a] focus on the question under which circumstances, i.e., parameter setting, a particle swarm will eventually converge. However, often the quality of the best found solution is not considered. Mostly, simplified, e.g., derandomized or one-dimensional, models of particle swarm optimization are analyzed. Runtime analyses, which take the solution quality into account, are mostly restricted to selected functions and very specific PSO approaches [PL07, SW08, Wit09]. However, the selection of a good or even optimal parameter setting strongly depends on the optimization problem to be solved. Although both convergence and runtime analyses provide guidelines for the parameter selection process, and standard parameter settings were derived [BK07], they do not offer any strict rules for choosing the PSO parameters for a specific practical application.

Parameter selection is a difficult task for other meta-heuristic optimzation approaches like evolutionary algorithms or ant colony optimization, too. Evolution strategies therefore optimize both the algorithmic parameters and the problem parameters during runtime, by, for instance, applying the $1/5$-th rule of Rechenberg [Rec73] or using mutative strategy parameter control [Rec94] (see Section 2.6.1).

In the subsequent section, existing adaptive particle swarm optimizers are briefly discussed. Afterwards, two novel adaptive particle swarm optimizers, the so-called *Multi-Swarm PSO with Migration (MPSO)* [JHW08] and *Particle Swarm Optimization with Velocity Adaptation* [HNW09] are presented.

5.1 Related Work

The existing adaptation mechanisms of particle swarm optimization can be classified into two categories: *Time-dependent adaptations* and *problem-dependent adaptations*.

The goal of most time-dependent adaptations is to modify the algorithmic parameters of a particle swarm optimizer such that initial exploration is gradually replaced by exploitation. However, problem-specific knowledge gathered during the optimization process is not taken into account. The adaptation is solely based on a time measure like the current number of iterations in relation to the maximum number of iterations allowed. In contrast, problem-dependent adaptations use problem-specific measures, e.g., the particles' success rates, for the adapation procedure. The target is to adjust the PSO algorithm and its parameters to the characteristics of the specific optimization problem under consideration.

Time-dependent Adaptations

In the literature, there exist several approaches for time-dependent modifications of the algorithmic PSO parameters. When introducing the inertia weight ω in 1998, Eberhart and Shi proposed to linearly decrease ω from $\omega = 1.4$ to $\omega = 0$ [SE98]. Thus, velocities tend to be larger at the beginning of the optimization than in the final iterations. Later, the same authors used a slightly shorter interval: The inertia weight was linearly decreased from $\omega = 0.9$ to $\omega = 0.4$ [SE99].

Ratnaweera et al. propose a similar strategy for the adaptation of the acceleration coefficients c_1 and c_2 [RHW04]. Again, the goal is to replace exploration by exploitation during the optimization process. Hence, the influence of a particle's private guide is gradually decreased, e.g., c_1 is reduced from an initial value of $c_1 = 2.5$ to $c_1 = 0.5$. The impact of a particle's local guide is, however, increased: c_2 is, for instance, raised from $c_2 = 0.5$ to $c_2 = 2.5$. The aim of these adaptations is to force particles to converge on the same solution in the later steps of the algorithm.

Besides inertia weight and acceleration coefficients, the structure of the neighborhood graph can be modified according to a time-dependent adaptation mechanism. The more neighbors a particle has, the faster its private guide is distributed among the other swarm members. If all particles are connected, each particle uses the same search space position as local guide, and the swarm often converges very fast. In order to enhance the exploration ability of a particle swarm, more sparsely connected neighborhood graphs, e.g., the ring topology in which each particle has exactly two neighbors, can be used[1].

In order to replace inital exploration by exploitation, both Suganthan [Sug99] and Richards and Ventura [RV03] suggest to adapt the particles' social network. Initially, a sparsely connected neighborhood graph like the ring topology is used. Then, step-by-step, new edges are added until the swarm is fully connected. As a second variant, Suganthan proposes to utilize the search space distances among the particles for the definition of the neighborhood graph: At the beginning of the optimization, only nearby particles are used for the evaluation of a particle's local guide. Gradually, more distant particles are included, until all particles share their information in the final steps of the PSO algorithm.

[1]See Section 2.2 for more details on neighborhood topologies. The fully connected graph and the ring topology are depicted in Figure 2.3 on Page 18.

Problem-dependent Adaptations

The adaptive approaches presented above adjust selected PSO parameters with respect to a time measure, usually the current number of iterations in relation to the total number of iterations used for the optimization run. However, the characteristics of the objective function are not taken into account. The following PSO variants adapt the algorithmic parameters considering problem-dependent measures, and hence aim at finding suitable parameters for the current problem to be solved.

As already mentioned, for evolution strategies[2] there exist several approaches to adapt the algorithmic parameters to the current problem, e.g., mutative strategy parameter control [Rec94]. Miranda and Fonseca combined concepts of evolution strategies and particle swarm optimization. The resulting adaptive optimization algorithm is called *Evolutionary Particle Swarm Optimization (EPSO)* [MF02a,MF02b]. In this adaptive PSO variant, each particle i maintains its personal values for the parameters ω, c_1 and c_2, which are adapted during the optimization process. The following PSO update equations are used [MF02a]:

$$\vec{v}_{i,t} = \omega_{i,t} \cdot \vec{v}_{i,t-1} + c_{1,i,t} \cdot (\vec{p}_{i,t-1} - \vec{x}_{i,t-1}) + c_{2,i,t} \cdot (\vec{l}^{*}_{i,t-1} - \vec{x}_{i,t-1}) \tag{5.1}$$

$$\vec{x}_{i,t} = \vec{x}_{i,t-1} + \vec{v}_{i,t} \tag{5.2}$$

where t is the iteration counter, and $\vec{l}^{*}_{i,t-1}$ is the mutated position of particle i's local guide $\vec{l}_{i,t-1}$. Mutation can for instance take place by adding a vector with normally distributed values to $\vec{l}_{i,t-1}$. Note that, compared to the standard PSO update equations shown in Equations (2.1) and (2.2), the stochastic components $\vec{r}_1$ and $\vec{r}_2$ were removed. $\omega_{i,t}$, $c_{1,i,t}$ and $c_{2,i,t}$ can either be scalar values as presented in Equation (5.1) [MF02a], or vectors [MF02b]. In the latter case, component-wise multiplication is used.

The parameters $\omega_{i,t}$, $c_{1,i,t}$ and $c_{2,i,t}$ can be mutated before updating a particle's position and velocity, similar to mutative strategy parameter control used in evolution strategies [MF02a]:

$$\begin{array}{rcl} \omega_{i,t} & = & \omega_{i,t-1} + \tau \cdot N(0,1) \\ c_{1,i,t} & = & c_{1,i,t-1} + \tau \cdot N(0,1) \\ c_{2,i,t} & = & c_{2,i,t-1} + \tau \cdot N(0,1) \end{array} \tag{5.3}$$

where $N(0,1)$ is drawn from the standard normal distribution, and τ is a so-called *learning parameter*, which can either be set to a fixed value or be treated as algorithmic parameter and undergo mutation. If vectors are used for the parameters, Equation (5.3) is adjusted so that normally distributed values are added to each component [MF02b]. Following the concept of evolutionary algorithms, a selection process concludes each iteration: Instead of moving each particle to its new position, the individuals that form the next iteration are chosen based on their solution quality.

[2]See Section 2.6.1 for more details on evolution strategies.

Due to the mutation of the algorithmic parameters and the selection procedure, particles with parameters that produced solutions of good quality are more likely to survive for the next generation while unsuccessful settings disappear. Like in evolution strategies, the aim is to learn appropriate parameters for the current problem to be solved during the optimization. Summarized, the EPSO algorithm repeats the following five steps in each iteration [MF02b]:

- *Replication:* Each particle is replicated r times.
- *Mutation:* Each new particle i mutates its parameters $\omega_{i,t}$, $c_{1,i,t}$, and $c_{2,i,t}$ according to Equation (5.3).
- *Reproduction/Movement:* Each particle (original or mutated) moves to a new position according to Equations (5.1) and (5.2).
- *Evaluation:* Each particle evaluates its fitness.
- *Selection:* The individuals of the next generation are selected based on their solution quality.

The EPSO algorithm was tested on commonly-used benchmark functions and on a practical application and yielded very good results [MF02a, MF02b].

Another example for PSO with problem-dependent adaptation is the *Efficient Multi-Objective PSO (EMOPSO)* of Toscano-Pulido et al. [PCSQ07]. EMOPSO uses the selection mechanisms of evolutionary algorithms for parameter adaptation. However, in contrast to EPSO, the algorithmic parameters are neither mutated nor recombined. Instead, all possible parameter values are specified at the beginning of the optimization, e.g., $\omega = 0$, $\omega = 0.5$, and $\omega = 1$. Each of these settings has a *fitness value* which is adapted during the optimization: The more non-dominated solutions a particular parameter value has produced, the higher is its fitness. With these definitions in mind, the values for ω, c_1, and c_2 are chosen based on their respective fitness values whenever a particle is updated.

While EPSO and EMOPSO try to adjust the continuous parameters ω, c_1 and c_2, Clerc proposes a PSO variant called *Tribes*, in which the number of particles and their social network is adapted during the optimization process [Cle03, Cle06b]. Based on the particles' performance, individuals are added to or removed from the swarm. All particles that were constructed at the same iteration are collected to form a so-called *tribe*. Inside a tribe, a densely (e.g., fully) connected neighborhood graph is used. In each iteration, the tribes are divided into *good tribes* and *bad tribes* according to specific rules, which take the performance of the particles into account. Each good tribe has to remove its worst particle. The neighborhood graph is adjusted so that the swarm is still connected. Each bad tribe generates a particle, and connects itself to it. As already mentioned, all particles constructed in the same iteration form a new tribe.

The adaptive Tribes algorithm deletes particles which are probably not important for the remaining search process in order to save function evaluations. It is assumed that good tribes can improve without their worst particles. On the other hand, bad tribes need more information because they seem to be stuck in local optima or are unable to converge. Therefore, bad tribes are connected to a newly generated tribe, which hopefully provides new directions for the search.

In the Tribes algorithm, the classical PSO equations as presented in Equations (2.1) and (2.2) were replaced. Instead, depending on a particle's quality, one of two so-called *pivot methods* is chosen [Cle03]. The algorithmic parameters of the pivot methods are adjusted according to the objective values of a particle's private and local guide. This way, they are indirectly adapted to the current problem to be solved, and no further user-specified parameters are needed.

Janson and Middendorf [JM05] propose a *Hierarchical Particle Swarm Optimizer* that uses a dynamic tree as neighborhood graph. In each iteration, the tree is restructured according to the particles' performance, which is meassured by taking the fitness values of their private guides into account. Good particles move towards the root of the tree, gaining more influence on other particles this way. Furthermore, different update rules can be used in different levels of the trees. As an additional time-dependent adaptation, Janson and Middendorf suggest to gradually decrease the tree's branching degree during the course of optimization.

5.2 Multi-Swarm PSO with Migration (MPSO)

The experimental study in the previous section showed that the algorithmic PSO parameters, such as the bound handling strategy, strongly influence particle swarm performance. Each parameter setting has its specific strengths and weaknesses, and is suited for a certain class of optimization problems. If the characteristics of the optimization problem are known beforehand, this problem-specific knowledge can be exploited for the manual adjustment of the algorithmic PSO parameters to the problem at hand by an expert. Parameter self-adaptation is useful if problem-specific knowledge is not available, hard to obtain, or hard to exploit (due to problem complexity, monetary costs, or time constraints, for instance).

The goal of *Multi-Swarm PSO with Migration (MPSO)* [JHW08], which is presented in this section, is to dynamically adapt the algorithmic PSO parameters to the given problem. This way, the practical application of particle swarm optimization is simplified. In contrast to other adaptive PSO algorithms, MPSO is a very general approach, which is able to adapt different kinds of parameters, such as binary, discrete, or continuous ones, to the given optimization problem. The algorithm is detailed in the next section. Experimental results are presented afterwards.

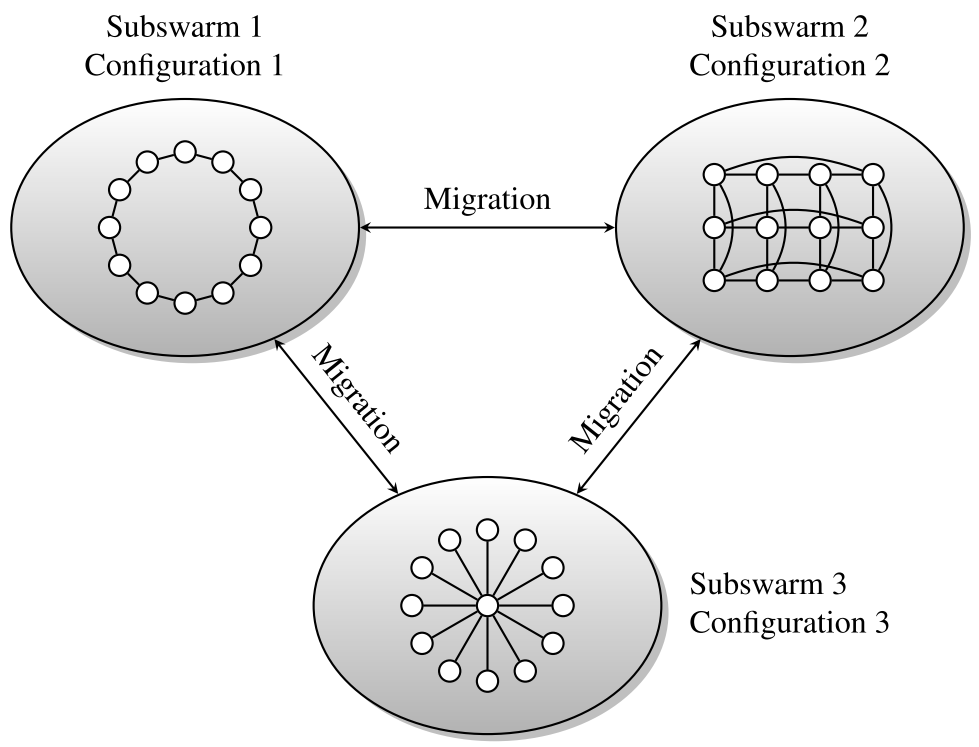


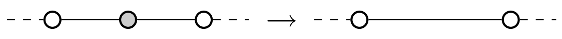
Figure 5.1: Adaptive Multi-Swarm PSO with Migration. The particle swarm is divided into several subswarms that cannot directly communicate with each other. From time to time, particle migration takes place.

5.2.1 The Algorithm

In Multi-Swarm PSO with Migration, the particle swarm is divided into a predefined number of competing subswarms that cannot directly communicate with each other. Each subswarm maintains its own parameter configuration, for instance, its own setting for the inertia weight, the acceleration coefficients, the bound handling strategy, and/or the neighborhood topology. From time to time, the performance of the subswarms is assessed according to specified criteria, and, based on this evaluation, particles are migrated from one subswarm to another. The goal is to increase the number of particles in promising subswarms, and to reduce the number of particles in subswarms with comparatively bad performance. Migrating particles keep their position, velocity, and their private guide, but they adopt the parameter configuration of the new subswarm. The algorithm is schematically illustrated in Figure 5.1.

The migration procedure consists of two steps: a particle has to be deleted from its old and added to its new subswarm. Different strategies are possible for this process, for example, the topologies of the involved subswarms can be rearranged every time a particle migrates. However, in this case, it can happen that the neighborhood of many particles is modified, and their search trajectories are disturbed. Therefore, particle insertion and deletion were designed such that, as far as possible, only a particle's immediate neighbors are affected. For the commonly-used fully connected graph and the ring and grid topologies, the following strategies can be applied:

- Fully connected neighborhood graph:
  - Deletion: Remove a particle and its communication links.
  - Insertion: Add a communication link from the inserted particle to each particle in the new subswarm.
- Ring topology:
  - Deletion: Remove a particle and its communication links. Connect its two neighbors with a new communication link.

  - Insertion: Insert the particle at a random position. Alternatively, insert the particle into a gap that arose from previous deletion processes (note that the gaps have to be logged in this case), or at a predefined position.
- Grid topology:
  - Deletion: Remove a particle and its communication links. Add two new communication links, from its top to its bottom neighbor, and from its left to its right neighbor.
  - Insertion: Insert the particle into a gap that arose from previous deletion processes. If no gaps are available, insert the particle at the end of the grid, by possibly expanding it in one dimension.

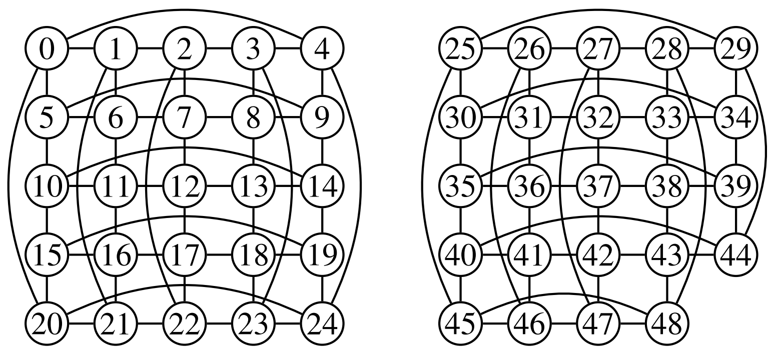
  These grid insertion and deletion procedures are used in the subsequent experiments. An example is given in Figure 5.2.

For the migration process, a particle has to be selected from the emitting subswarm. Several strategies are possible, e.g., the worst or the best particle can be chosen. For simplicity, and due to the assumption that this choice does not greatly influence particle swarm behavior, the migrating particle is drawn uniformly at random from the ceding subswarm.

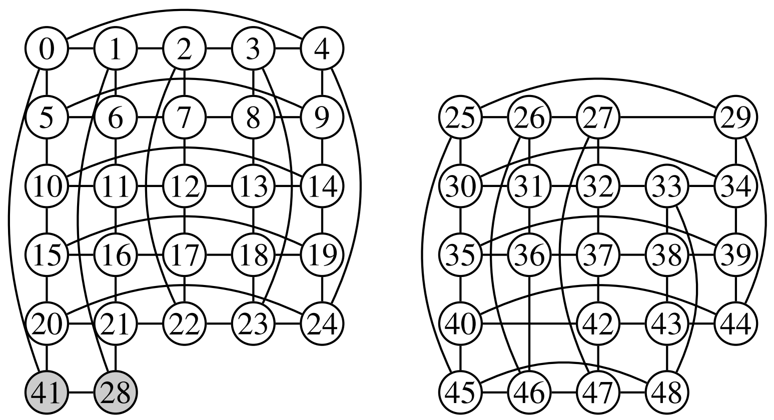
Based on this general MPSO framework, two adaptive particle swarm optimizers were developed [JHW08]. They are presented in the following.

MPSO-1

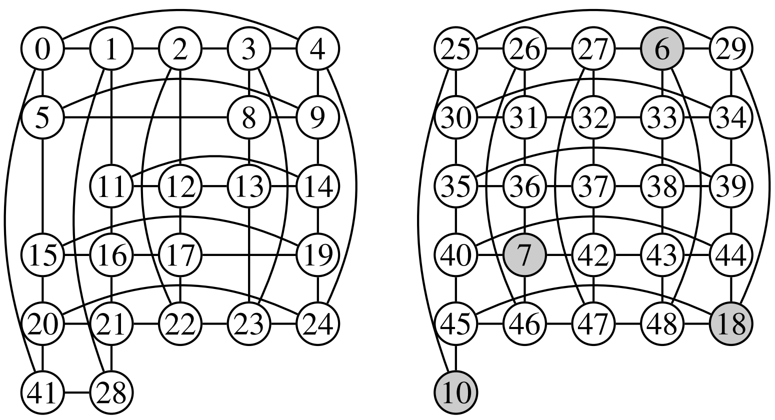
MPSO-1 is a rather simple, yet effective realization of the MPSO concept. The algorithm was specifically developed with the design goal simplicity in mind: easy to



(a) Initialization: 49 particles are divided into two subswarms of similar size.



(b) Particles 41 and 28 are deleted from the subswarm on the right-hand side (in this order). As the left subswarm has no gaps in the grid topology yet, they are added at the end of the grid.



(c) The left subswarm lost particles 6, 7, 18, and 10 in the next iterations. First, the gaps are filled with particles 6 and 7. The remaining particles are added at the end of the grid.

Figure 5.2: Example of particle migration in a grid topology.

understand, easy to implement, and easy to apply, which in this case means that only a single new parameter is introduced.

In MPSO-1, each subswarm memorizes as its fitness value the best objective value that was ever carried as private guide by one of its members. In order to increase the number of particles in subswarms with better fitness, particle migration periodically takes place every k_1 iterations, where k_1 is the user-defined parameter. Every k_1 iterations, a random particle migrates from the subswarm with worst fitness to one with best fitness.

MPSO-2

In this realization, the concept of penalty points is introduced. Instead of periodically triggering migration after a certain number of iterations, penalty points are assigned to subswarms that perform bad on a specified criterion. As soon as the amount of a subswarm's penalty points exceeds a user-defined limit P_{limit}, particle migration takes place.

Let $f(\vec{y}_{j,t})$ be the best objective value observed by subswarm j until iteration t, and let k_2 and C_{max} be user-defined parameters. Then, the criterion value of subswarm j is defined as:

$$
\begin{aligned}
crit_j &= \begin{cases} f(\vec{y}_{j,t-k_2})/f(\vec{y}_{j,t}) - 1 & \text{if } f(\vec{y}_{j,t-k_2}) > 0 \text{ and } f(\vec{y}_{j,t}) > 0 \\ f(\vec{y}_{j,t})/f(\vec{y}_{j,t-k_2}) - 1 & \text{if } f(\vec{y}_{j,t-k_2}) < 0 \text{ and } f(\vec{y}_{j,t}) < 0 \\ 0 & \text{otherwise} \quad . \end{cases} \\
criterion_j &= \min\{crit_j, C_{max}\}
\end{aligned}
$$

Obviously, $0 \leq criterion_j \leq C_{max}$ holds. The greater $criterion_j$, the more progress was achieved by subswarm j during the last k_2 iterations. Subswarms that make more progress than their competitors are rewarded with additional particles, while the others have to give up some of their particles. The criterion value is bounded by C_{max}, usually set to a very small value, which means that the performance of all subswarms that strongly improved their best found objective value in the last period is considered as equally good.

For the computation of the amount of penalty points that is added to a subswarm's penalty record, the criterion values of all subswarms are compared pairwise. If the ratio $criterion_i/criterion_j$ is greater than a user-defined limit $P_{ratio} > 1$, a penalty point is added to subswarm j.

In each iteration, the subswarms' criterion values are computed, and their penalty records are updated. In order to determine which subswarms are favored for the insertion of emitted particles, all subswarms are sorted according to their current criterion value. Then, if applicable, a particle is migrated from every subswarm whose penalty record exceeds the limit P_{limit} to another subswarm in this order. The penalty records of the emitting subswarms are reset.

5.2.2 Experimental Results

The aim of *Multi-Swarm PSO with Migration* is to provide an algorithm that is able to choose an appropriate parameter configuration for the problem at hand during runtime, from a set of predefined configurations. While MPSO-1 was designed with simplicity in mind, MPSO-2 uses penalty points to control the parameter adaptation.

The goals of the experimental analysis can be formulated as follows:

- Is MPSO able to adapt to a promising parameter configuration?
- Are there significant performance differences between MPSO-1 and MPSO-2?

In order to find out whether MPSO is able to adapt to a parameter configuration that is suited for the given problem, two complementing bound handling strategies were selected for the subswarms: Reflect-Z and Hyperbolic. Although, in the experimental investigation presented in Section 4.4.2, Hyperbolic showed very good performance on most of the 100-dimensional problems, it had severe difficulties with Schwefel, f9, and f10. On these three functions, Reflect-Z performed best. Hyperbolic and Reflect-Z already perfectly cover the given benchmark set: On each function, either Reflect-Z or Hyperbolic provided the best average results (see Tables B.16 and B.17 in the appendix on page 195). Hence, a third configuration is not needed.

Complete PSO algorithms such as Stereotyping (see Section 2.2.3) and Ranked FIPS (see Section 2.2.5) can also be used as subswarm configurations. In an experimental study using complete algorithms as subswarms, the inclusion of a third parameter configuration often did not pay off [JHW08]. With each additional parameter set, the size of the subswarms decreases, which means that less ressources are spent per configuration. Therefore, and due to the fact that there is no obvious third candidate, only two bound handling strategies were used as subswarm configurations in this study.

The PSO parameters were set to standard values according to Table 4.1 on page 89. The population of 49 particles was split into two nearly equally-sized subswarms, as shown in Figure 5.2a. Half-diff velocity initialization was used. The test functions are given in Section 4.2. The performance of MPSO-1 and MPSO-2 was investigated on both traditional benchmarks such as Sphere, Rosenbrock, and Rastrigin, and on CEC 2005 benchmarks [SHL$^+$05]. The problem dimensionality was set to $n = 100$. When solving 100-dimensional problems, significant performance differences were observed between the selected bound handling methods (see Chapter 4), which allows the examination of the first experimental goal. For MPSO-1, k_1 was set to $k_1 = n = 100$, if not stated otherwise. The parameters of MPSO-2 were set to $k_2 = 200$, $C_{max} = 0.01$, $P_{limit} = n/2 = 50$, and $P_{ratio} = 2$.

Both MPSO-1 and MPSO-2 were mostly able to adapt the algorithmic PSO parameters to the current problem at hand. Table 5.1 shows that often the sample mean of the final objective values achieved by *Multi-Swarm PSO with Migration* settled between the means of its components (i.e., the strategies used in the subswarms), as expected, especially when solving CEC 2005 benchmarks. For instance, on Rastrigin and f2, the performance of MPSO approaches Hyperbolic, while on Schwefel and f9 it approaches Reflect-Z, as illustrated in Figures 5.3 and 5.4. Note that an adaptive algorithm is usually not supposed to produce better results than its constituents, but to more reliably provide satisfactory results on a broad variety of different problems. The main benefit is the reduction of the need of manual parameter adjustment to the current problem at hand.

Although both algorithms are based on the same framework, the swarm behavior differs. In MPSO-1, sooner or later all particles migrated to the same subswarm. Figure 5.5 shows how often the final number of particles in the first subswarm (in this case the one with Reflect-Z bound handling) reached certain values. Note that a logarithmic scale is used in this plot. In most runs, namely in 1,787 out of 1,800 runs, the first subswarm was either full (49 particles) or empty (0 particles). Only very few exceptions exist. Mostly, the better-performing bound handling strategy was chosen, e.g., Hyperbolic on Rastrigin and f2, and Reflect-Z on Schwefel and f9.

A different behavior was observed in MPSO-2. Due to the fact that not performance but the subswarm's ability to improve the current best solution is rewarded, particles do not migrate anymore if both subswarms converged. This is visible in the broad variety of final subswarm sizes observed with MPSO-2 (see Figure 5.6). However, note that again a logarithmic scale is used in this plot. In most runs, the final subswarm size was either 0, 25, or 49.

On some functions, for instance on f2, both Hyperbolic and Reflect-Z steadily improve the best found solution, as depicted in Figure 5.4 (bottom). Since both strategies are equally successful considering the criteria of MPSO-2, particles do not migrate. Figure 5.7 shows the frequency of final subswarm sizes for some selected benchmarks. As already mentioned, when solving f2, particles did not migrate, and hence, the final subswarm size of the first subswarm was still 25 at the end of the optimization in each run.

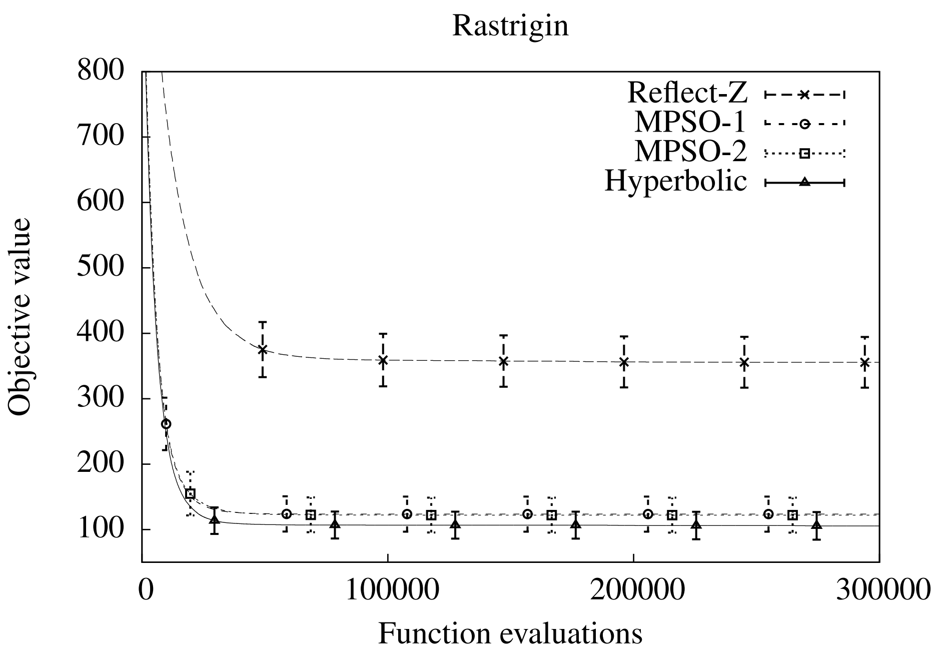
When solving Rastrigin, particles often rapidely converge. Hence, a broad spectrum of final subswarm sized was observed for this function (see Figure 5.7). The sample mean and the median of the final subswarm size of the first subswarm were 19.58 and 24, respectively. The standard deviation of the sample mean was relatively high (approximately 12.52).

On Schwefel and f9, Reflect-Z mostly produced better results than Hyperbolic velocity update in the previous experiments presented in Section 4.4.2. Figure 5.4 shows that in particular on f9, the solution quality can be still improved in the later stages of the optimization when using Reflect-Z bound handling. Accordingly, in most runs all particles migrated to the respective subswarm (see Figure 5.7).

Despite the different swarm behavior of MPSO-1 and MPSO-2, the performance of the two algorithms was mostly comparable on the tested benchmark functions. The obtained average objective values were similar (see Table 5.1), and significant

Table 5.1: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) for the 100-dimensional benchmarks. The best objective values are presented together with the function name.

	Sphere (0)	Rosenbrock (0)
Hyperbolic	5.9506e-06±1.6691e-07 (8.412e-07)	**152.94±9.4227 (47.488)**
Reflect-Z	*6.0293e-06±1.7838e-07 (8.9898e-07)*	176.34±10.406 (52.445)
MPSO-1	**1.1077e-06±7.5428e-08 (3.8014e-07)**	*187.1±9.5036 (47.896)*
MPSO-2	1.4448e-06±7.2735e-08 (3.6657e-07)	170.05±9.1775 (46.253)
	Ackley (0)	Griewank (0)
Hyperbolic	**0.37429±0.12799 (0.64502)**	**3.5747e-03±1.3876e-03 (6.9929e-03)**
Reflect-Z	1.6946±0.13572 (0.68397)	3.8688e-03±1.6439e-03 (8.2849e-03)
MPSO-1	1.7576±0.12814 (0.64582)	*8.5606e-03±4.9031e-03 (0.02471)*
MPSO-2	*1.9617±0.097992 (0.49386)*	4.1591e-03±2.7326e-03 (0.013772)
	Rastrigin (0)	Schwefel (≈ -41898.3)
Hyperbolic	**105.46±4.1927 (21.13)**	*-24848±333.63 (1681.4)*
Reflect-Z	*355.61±7.7081 (38.847)*	**-30569±305.32 (1538.7)**
MPSO-1	123.61±5.3266 (26.845)	-29715±337.31 (1700)
MPSO-2	122.04±5.2901 (26.661)	-29684±378.53 (1907.7)
	f1 (-450)	f2 (-450)
Hyperbolic	**-450 (0)**	**3348.3±123.56 (622.7)**
Reflect-Z	**-450 (0)**	*23506±1217.5 (6135.7)*
MPSO-1	**-450 (0)**	3689.3±147.04 (741.06)
MPSO-2	**-450 (0)**	4012.3±148.64 (749.1)
	f3 (-450)	f5 (-310)
Hyperbolic	12062000±446790 (2251700)	27956±572.45 (2885)
Reflect-Z	*3.613e+07±1841200 (9279000)*	**27456±697.98 (3517.7)**
MPSO-1	12706000±446140 (2248400)	29708±673.38 (3393.7)
MPSO-2	**11726000±565350 (2849200)**	*30174±587.33 (2960)*
	f6 (390)	f8 (-140)
Hyperbolic	570.68±8.9378 (45.045)	**-118.71±6.4444e-03 (0.032478)**
Reflect-Z	*583.66±10.556 (53.201)*	**-118.71±5.8188e-03 (0.029325)**
MPSO-1	**567.62±10.023 (50.515)**	**-118.71±6.1638e-03 (0.031064)**
MPSO-2	571.89±10.174 (51.277)	**-118.71±6.393e-03 (0.032219)**
	f9 (-330)	f10 (-330)
Hyperbolic	*264.95±9.7299 (49.036)*	*478.7±15.556 (78.397)*
Reflect-Z	**3.579±10.44 (52.615)**	**8.2564±12.608 (63.541)**
MPSO-1	68.915±13.298 (67.02)	101.54±17.918 (90.302)
MPSO-2	81.999±13.201 (66.528)	100.68±18.914 (95.32)
	f11 (-460)	f12 (90)
Hyperbolic	**214.65±1.5314 (7.7177)**	**173200±15811 (79686)**
Reflect-Z	*218.24±1.2826 (6.4642)*	*599230±54540 (274870)*
MPSO-1	217.79±1.4298 (7.2058)	229570±22491 (113350)
MPSO-2	216.53±1.503 (7.5748)	227710±26306 (132570)
	f13 (-130)	f14 (-300)
Hyperbolic	**-102.77±0.93404 (4.7074)**	**-254.18±0.11919 (0.60069)**
Reflect-Z	*-65.84±2.7268 (13.742)*	*-253.44±0.079587 (0.4011)*
MPSO-1	-87.593±1.8549 (9.3484)	-253.88±0.10943 (0.55149)
MPSO-2	-88.907±1.6569 (8.3506)	-253.85±0.14126 (0.7119)



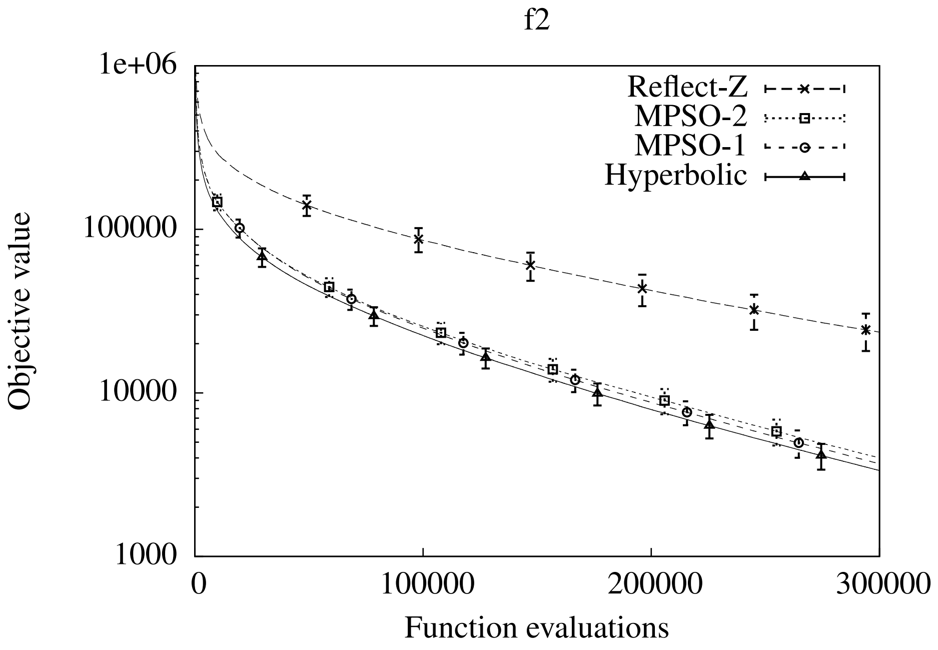
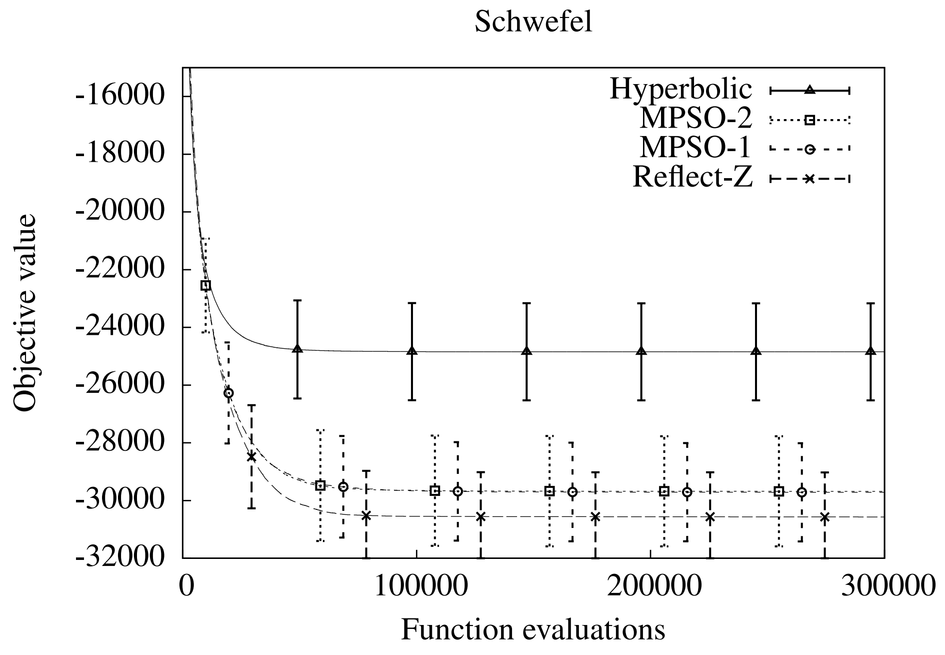


Figure 5.3: On many functions, both MPSO-1 and MPSO-2 were able to adapt to the best suited parameter configuration. For instance, on Rastrigin and f2, the performance of Hyperbolic was approached. In the plot, average values and standard deviations (vertical bars) are shown.



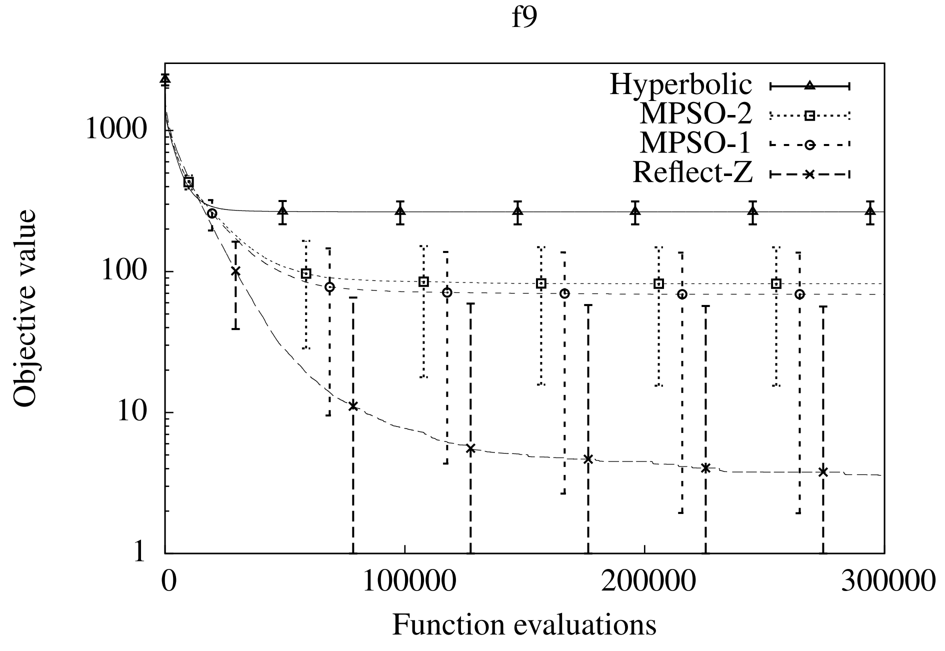


Figure 5.4: Mostly, the solution quality of MPSO-1 and MPSO-2 settled between the ones of its two constituent subswarm configurations, notwithstanding if Hyperbolic or Reflect-Z produced better results on the given benchmark problem (compare also to Figure 5.3). Note that a logarithmic scale is used in the second plot, which makes the standard deviations of the algorithms with better performance appear larger.

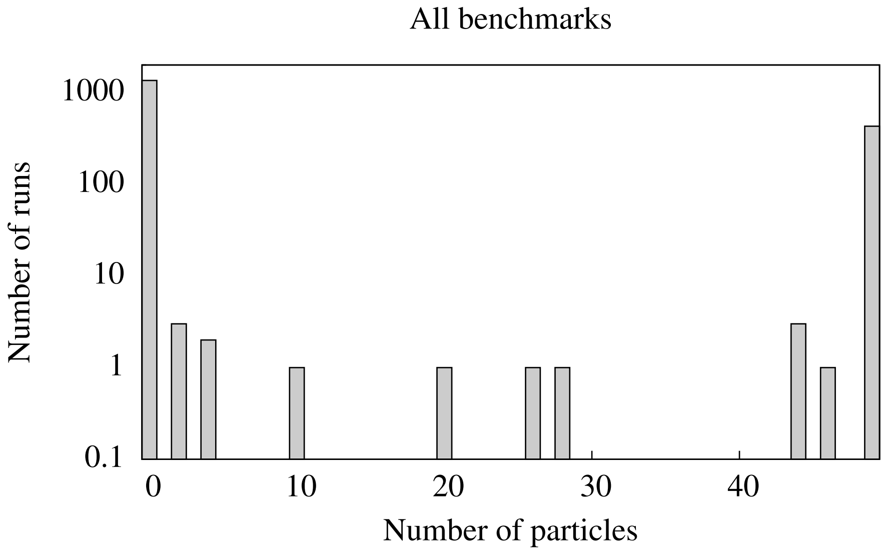


Figure 5.5: The histogram shows how frequently the final number of particles in the first subswarm (Reflect-Z bound handling) reached certain values. The data of all runs was used to generate this histogram. In 1,787 out of a total number of 1,800 runs, all particles belonged to the same subswarm at the end of the optimization. Note that a logarithmic scale is used in this plot.

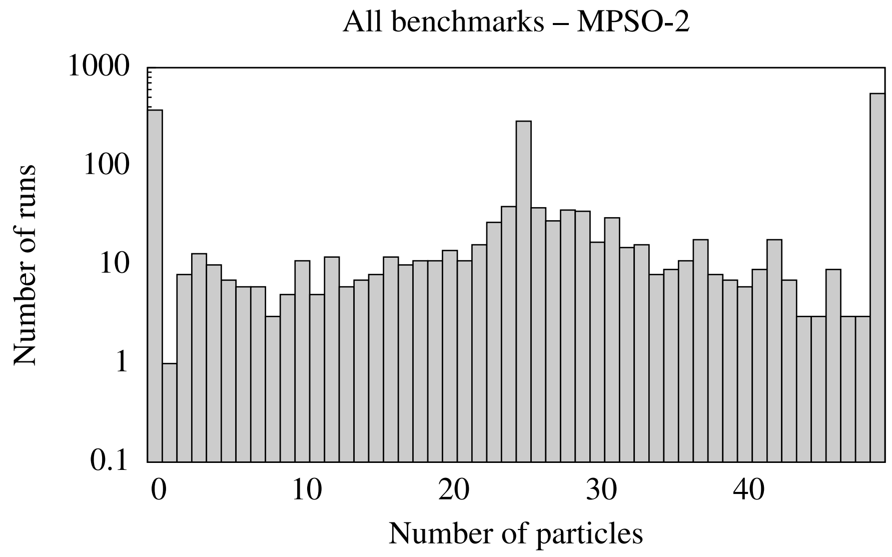


Figure 5.6: The histogram shows how frequently the final number of particles in the first subswarm (Reflect-Z bound handling) reached certain values. The data of all runs was used to generate this histogram. Note that a logarithmic scale is used in this plot.

performance differences were only observed in four cases, which is almost negligible due to the high number of statistical tests performed in this setup. The results of the Wilcoxon rank sum test are summarized in Table 5.2.

Summarized, both MPSO-1 and MPSO-2 were often able to adapt to the most promising parameter configuration. The obtained final solution quality mostly settled between the one of its components (i.e., the algorithms used in the subswarms), which is the minimum requirement of an adaptive algorithm. Often, MPSO almost achieved the performance of the best predefined parameter configuration by means of adaptation. Compared with each other, MPSO-1 and MPSO-2 showed differences in their swarm behavior, however, solutions of similar quality were achieved in this experimental investigation. Nevertheless, the efforts spent on the design of MPSO-2 might pay of in other scenarios, for instance, if complete PSO algorithms are used as subswarm configurations [JHW08].

In MPSO-1, particle migration takes place every k_1 iterations. In the following, the impact of the parameter k_1 on the swarm's solution quality is briefly discussed. Experiments were conducted with $k_1 = 5, 50, 100, 200, 500$. Sample means, corresponding 95% confidence intervals and standard deviations can be found in Appendix B.5 in Tables B.24 and B.25. The summarized results of the Wilcoxon rank sum test are presented in Table 5.3.

The experimental results show that both too small and too large adaptation intervals

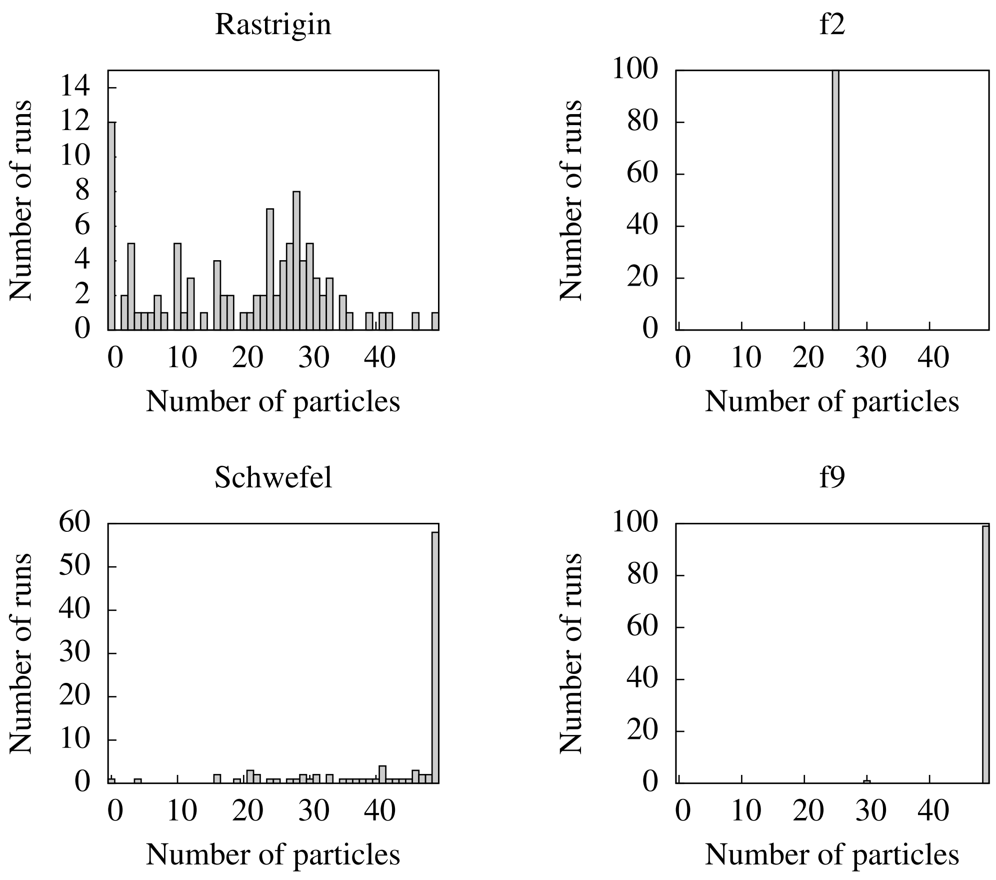


Figure 5.7: The histograms show how frequently the final number of particles in the first subswarm (Reflect-Z bound handling) reached certain values on four selected test functions, using MPSO-2.

Table 5.2: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows on which benchmarks A performed significantly better than B.

	Hyperbolic	Reflect-Z	MPSO-1	MPSO-2
Hyperbolic	{}	{Ro, Ack, Ra, f2, f3, f11, f12, f13, f14}	{Ro, Ack, Ra, f2, f5, f11, f12, f13, f14}	{Ro, Ack, Ra, f2, f5, f12, f13, f14}
Reflect-Z	{Schw, f9, f10}	{}	{Schw, f5, f9, f10}	{Ack, Schw, f5, f9, f10}
MPSO-1	{Sph, Grie, Schw, f9, f10}	{Sph, Grie, Ra, f2, f3, f12, f13, f14}	{}	{Sph, Ack, f2}
MPSO-2	{Sph, Grie, Schw, f9, f10}	{Sph, Grie, Ra, f2, f3, f12, f13, f14}	{f3}	{}

Table 5.3: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows on which benchmarks A performed significantly better than B.

	1	2	3	4	5
MPSO-1-5 (1)	{}	{Ack, f5, f13}	{Ro, Ack, f3, f5, f13, f14}	{Ack, Grie, f2, f3, f5, f13, f14}	{Sph, Ack, Grie, Ra, f2, f5, f13, f14}
MPSO-1-50 (2)	{Schw, f9, f10}	{}	{Ro}	{Ack, Grie, f9, f13, f14}	{Sph, Ack, Grie, f2, f13}
MPSO-1-100 (3)	{Schw, f9, f10}	{}	{}	{f9}	{Sph, Ack, f13}
MPSO-1-200 (4)	{Schw, f9, f10}	{}	{}	{}	{Sph}
MPSO-1-500 (5)	{Schw, f9, f10}	{}	{Ro}	{}	{}

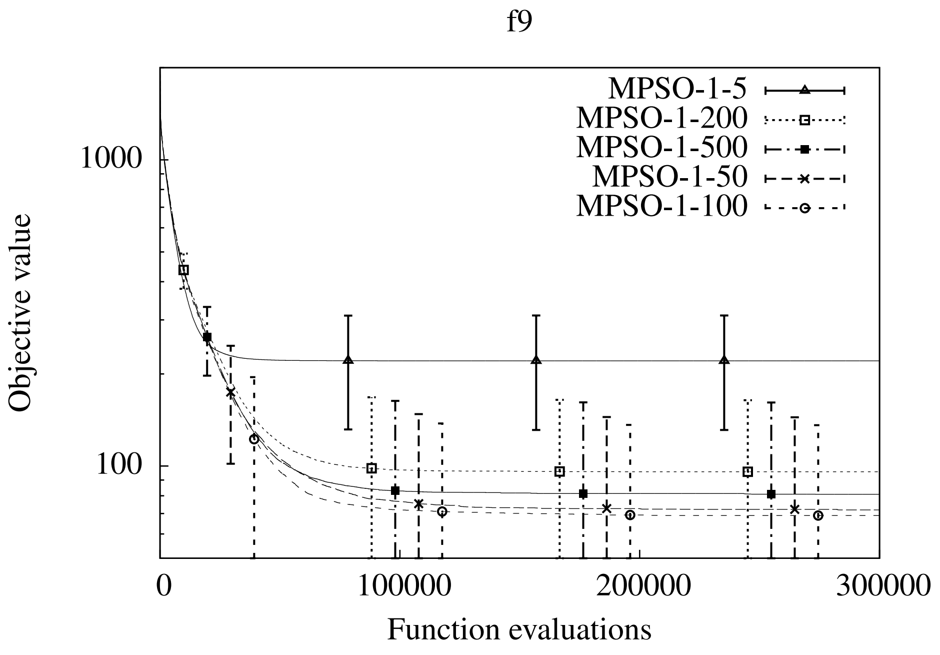
can deteriorate the performance of MPSO-1. For instance, a small adaptation interval of $k_1 = 5$ was disadvantegous for Schwefel, f9, and f10 (see Table 5.3). In these cases, the sample means were comparatively high (Tables B.24 and B.25) so that in general larger adaptation intervals are recommended. When using a small adaptation interval, e.g., $k_1 = 5$ (MPSO-1-5), particles rapidly migrate to the subswarm that it most successful at the beginning of the optimization. This can lead to premature convergence, as shown in the convergence plot for f9 in Figure 5.8 (top). On the other hand, if the selected strategy is also successful in the later steps of the optimization, ressources are effectively used from the beginning, and not wasted for unsuccessful configurations. For instance, $k_1 = 5$ worked fine for f13 as depicted in Figure 5.8 (bottom).

Although the usage of very high adaptation intervals ($k_1 = 200$ and $k_2 = 500$) led to significant performance losses in some cases (e.g., Ackley and f13), the obtained sample means are still acceptable. The sample means obtained with $k_1 = 50, 100, 200, 500$ are similar for most benchmarks (see Tables B.24 and B.25). Summarized, in this experimental investigation the selection of the newly introduced parameter k_1 did not greatly influence particle swarm performance, as long as the adaptation interval is not chosen very small ($k_1 = 5$) or very high ($k_1 = 500$).

5.3 Particle Swarm Optimization with Velocity Adaptation

Multi-Swarm PSO with Migration can be used to adapt the particles' bound handling strategy to the problem at hand. The algorithm is particularly beneficial if the characteristics of the problem are unknown beforehand. Instead of parameter adaptation, approaches that reduce or even eliminate the effect of certain PSO parameters on particle swarm performance are useful as an alternative. In the theoretical analysis presented in Chapter 3, it was proved that the selected bound handling strategy strongly influences initial particle swarm behavior when solving high-dimensional problems. The complementing experimental study (Chapter 4) showed that these effects are often also visible in the final solution quality provided by a PSO algorithm. As a solution to these observed effects, the algorithm presented in this section was developed to reduce the impact of bound handling on particle swarm performance. It is based on the theoretical results presented in Section 3.4, and was specifically designed for higher-dimensional problems.

The analysis presented in Section 3.4 showed that the probability that a particle becomes infeasible heavily depends on the length of its velocity vector. By using a simplified PSO model, it was proved that the probability of a particle becoming infeasible is constant if velocities are, for instance, chosen uniformly at random in $[-\frac{r}{n}, \frac{r}{n}]^n$. However, the usage of such small velocities can prevent search space ex-



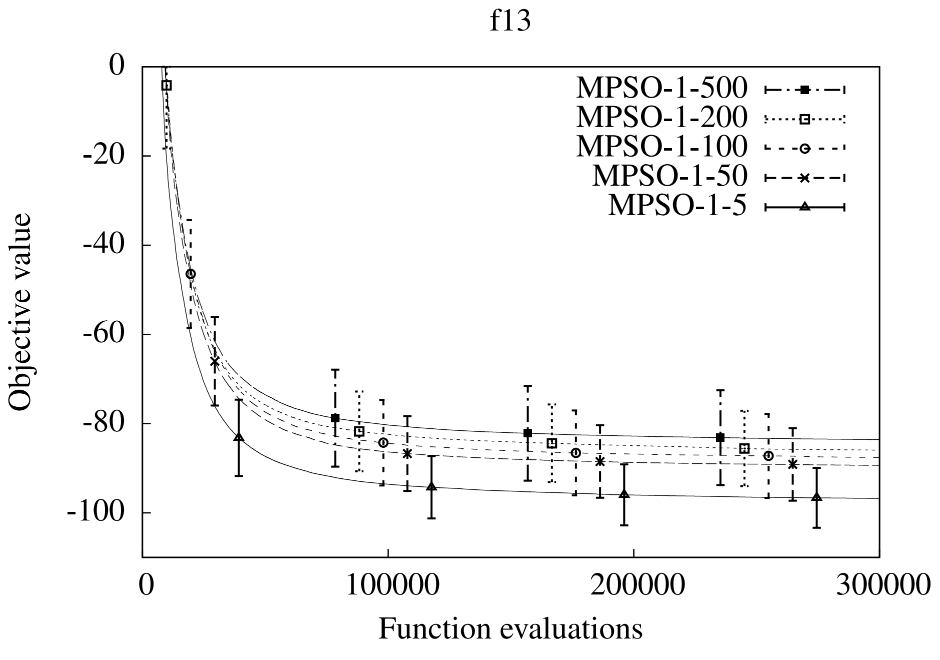


Figure 5.8: Convergence plots for the MPSO-1 algorithm with different values of k_1. E.g., MPSO-1-5 means that MPSO-1 was used with $k_1 = 5$. In this experimental investigation, good results were obtained with $k_1 = 50, 100, 200, 500$ for a broad variety of different benchmarks. As long as not chosen too small or too large, the selection of k_1 did not greatly influence the performance of MPSO-1.

ploration. In contrast, large velocities lead to better exploration capabilities of the particle swarm, at the expense of an increased probability that particles become infeasible.

In *Particle Swarm Optimization with Velocity Adaptation* [HNW09], particle velocities are adapted to the search process by analyzing the success of the particle swarm. The adaptation procedure is based on the 1/5-rule of Rechenberg [Rec73].

Particle velocities have strong impact on particle swarm behavior and particle swarm performance. The inclusion of a constant componentwise maximum particle velocity, also denoted as *velocity clamping*, was often found to improve the solution quality of a PSO algorithm [ES00]. Cui et al. [CZS06, CCZ09] propose to use a dynamic threshold for particle velocities. Their approach is inspired by evolutionary programming, but does, however, not take the swarm's success into account when updating maximum particle velocities.

In order to accelerate the convergence speed of a particle swarm, which can be useful if function evaluations are extremely expensive, Fan [Fan02] suggests to deterministically decrease particle velocities during the course of optimization from a given starting value $V_{max,d}$ (for the d-th component of a particle's velocity vector) to zero:

$$V_{max,t,d} = (1 - (t/T)^h) \cdot V_{max,d}$$

where t is the current time step, h is a user-defined parameter, and T is the maximum number of time steps. This approach does not take the swarm behavior into account either, and can therefore be categorized as a time-dependent adaptation.

Takahama and Sakei [TS06] extended their ε-constrained PSO with a componentwise adaptive maximum particle velocity. In their algorithm, particles are divided into subswarms. The performance of the subswarms is assessed with respect to the number of feasible particles. In each iteration, the maximum velocity of the worst subswarm is modified such that it approaches the maximum velocity of the best subswarm.

Fourie and Groenwold [FG02] adapt the maximum particle velocity based on the particles' success. Whenever the best found solution was not improved during a certain period of time, the maximum particle velocity is decreased:

$$V_{max,t,d} = \beta \cdot V_{max,t-1,d}$$

where $0 < \beta < 1$. This approach resembles *PSO with Velocity Adaptation*. There are, however, some major differences: (1) Fourie and Groenwold never increase particle velocities again, (2) their maximum velocity is altered and applied componentwise, and (3) they only take the best found solution into account for determining the swarm's success.

Particle Swarm Optimization with Velocity Adaptation is detailed in the subsequent section. Afterwards, experimental results are discussed.

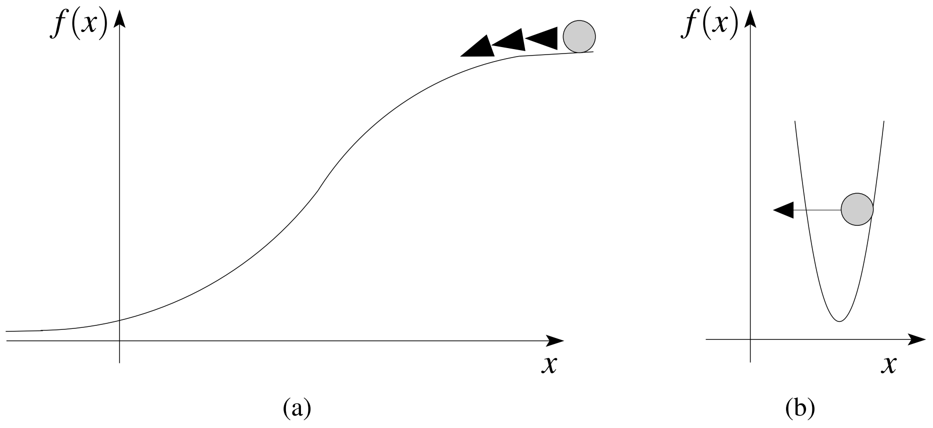


Figure 5.9: Illustration of the velocity adaptation strategy. Figure (a) depicts a situation in which large velocities are advantageous due to the fact that (often computationally expensive) function evaluations can be saved. Figure (b) shows a situation in which smaller velocities are necessary in order to approach the local optimum.

5.3.1 The Algorithm

The theoretical analysis presented in Section 3.4 showed how the probability that particles become infeasible depends on the intervals from which the particle velocities are chosen. The smaller a particle's velocity, the less likely it becomes infeasible. However, too small velocities lead to stagnation and a reduced explorative behavior of the particle swarm. For search space exploration, large velocities are needed.

In *Particle Swarm Optimization with Velocity Adaptation*, the particles' velocities are dynamically adapted to the search progress. The velocity adaptation mechanism is conceptually similar to the $1/5$-rule of Rechenberg [Rec73], which was introduced in the field of evolution strategies.The idea is to increase particle velocities as long as the best found solutions are constantly improved, and to decrease velocities if the swarm is currently not successful. If improvements are achieved repeatedly, the situation may be similar to the one depicted in Figure 5.9a. In this case, larger velocities are beneficial due to the fact that (often computationally expensive) function evaluations can be saved. On the other hand, if the swarm is not successful, it possibly overshoots local optima, as shown in Figure 5.9b. In such a situation, smaller steps can lead to further improvements. In order to adapt the particles' velocities to the optimization process, an adaptive *particle step size*, denoted as l_v (length of velocity vector), is introduced. Whenever a particle updates its velocity, it is scaled such that its absolute value is exactly l_v.

For the adaptation of the step size l_v, each particle's success is measured according to the following definition:

Definition 5.1 (Success)**.** *A particle i is called* successful *in iteration t if* $f(\vec{x}_{i,t}) <$

$f(\vec{p}_{i,t-1})$, where f is the objective function to be minimized, and $\vec{x}_{i,t}$ and $\vec{p}_{i,t}$ are the particle's current position and private guide, respectively. If $f(\vec{x}_{i,t}) = f(\vec{p}_{i,t-1})$, the particle is called successful with a probability of 1/2.

A particle is called successful if and only if its private guide is updated to its current position.

PSO with velocity adaptation is detailed in Algorithm 5.1. Let *SuccessCounter* denote the total number of successes achieved by m particles in the latest n iterations. Note that $SuccessCounter \le n \cdot m$ holds. Every n iterations, the success rate is updated to

$$SuccessRate = \frac{SuccessCounter}{n \cdot m} \; .$$

If the success rate exceeds a given threshold ρ, $0 \le \rho \le 1$, the step size l_v is doubled, otherwise it is halved. In accordance to the 1/5-rule introduced by Rechenberg [Rec73], the threshold can be set to $\rho = 0.2$. Different settings are analyzed in the subsequent experimental study.

In some PSO algorithms, a componentwise maximum particle velocity is used. In the presented PSO with velocity adaptation, velocity vectors are scaled such that their absolute values are equal to the current step size l_v. Due to the scaling strategy, the search direction is not modified if a particle's velocity exceeds its limits. Velocities are not only reduced if they exceed a specified maximum velocity, but they are also scaled up if they are too small. This way, the adaptation process controls the exploration and exploitation behavior of the particle swarm, in contrast to situations in which only a maximum velocity is used. Experimentation (presented below) confirms that the introduced *up- and down-scaling* mechanism is important for the success of the new algorithm.

5.3.2 Experimental Results

In order to analyze the new algorithm, a variety of experiments was performed. The main goals were to compare PSO with velocity adaptation to a standard PSO algorithm, and to analyze the impact of the newly introduced parameters on the algorithm's performance.

Experiment 1: Comparison with a Standard PSO

The first experiment focuses on two questions:

- Are there significant performance differences between PSO with velocity adaptation and a standard PSO?
- Is the new algorithm less sensitive to the choice of the bound handling strategy than a standard particle swarm optimizer?

Algorithm 5.1 PSO with velocity adaptation

```
Require: Objective function $f : \mathcal{S} \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, $\rho \in [0,1]$, Initial step size $l$
 1: Initialize particle positions, velocities, and private guides
 2: Initialize neighborhood graph
 3: $SuccessCounter \leftarrow 0$
 4: $l_v \leftarrow l$
 5: $t \leftarrow 0$
 6: for each particle $i$ ($i = 1, \ldots, m$) do
 7:     Scale initial velocity $\vec{v}_{i,0}$ such that $|\vec{v}_{i,0}| = l_v$
 8: end for
 9: repeat
10:     $t \leftarrow t + 1$
11:     for each particle $i$ ($i = 1, \ldots, m$) do
12:         Velocity update according to Eq. (2.1):
                $\vec{v}_{i,t} = \omega \cdot \vec{v}_{i,t-1} + c_1 \cdot \vec{r}_{i,t,1} \odot (\vec{p}_{i,t-1} - \vec{x}_{i,t-1}) + c_2 \cdot \vec{r}_{i,t,2} \odot (\vec{l}_{i,t-1} - \vec{x}_{i,t-1})$
13:         Scale velocity $\vec{v}_{i,t}$ such that $|\vec{v}_{i,t}| = l_v$
14:         Position update according to Eq. (2.2):    $\vec{x}_{i,t} = \vec{x}_{i,t-1} + \vec{v}_{i,t}$
15:     end for
16:     for each particle i ($i = 1, \ldots, m$) do {Private guide update}
17:         if (success($\vec{x}_{i,t}, \vec{p}_{i,t-1}$)) then
18:             $\vec{p}_{i,t} \leftarrow \vec{x}_{i,t}$
19:             $SuccessCounter \leftarrow SuccessCounter + 1$
20:         end if
21:     end for
22:     if $t \mod n = 0$ then {Adaptation of $l_v$}
23:         $SuccessRate \leftarrow \frac{SuccessCounter}{n \cdot m}$
24:         if $SuccessRate > \rho$ then
25:             $l_v \leftarrow 2 \cdot l_v$
26:         else
27:             $l_v \leftarrow l_v / 2$
28:         end if
29:         $SuccessCounter \leftarrow 0$
30:     end if
31: until termination criterion met
```

For the experimental study, the same benchmarks as in the previous section and in Chapter 4 were used. They are outlined in Section 4.2. PSO with velocity adaptation was compared to a standard PSO algorithm similar to the one of Bratton and Kennedy [BK07], as described in detail on page 88 in Section 4.1. For both approaches, the parameter settings summarized in Table 5.4 was used. Velocity clamping, which means that the particle velocities are componentwise limited by a minimum and maximum value, was often found to improve the solution quality obtained by particle swarm optimization (e.g., [ES00], and own experiments). In order to provide fair comparison, velocity clamping was introduced in the standard particle swarm optimizer for the following experimental investigations. If $\mathcal{S} = [lb_1, ub_1] \times \cdots \times [lb_n, ub_n]$ is the search space of the optimization problem, the d-th component of each particle's velocity vector is restricted by $(ub_d - lb_d)/2$. Note that, of course, this static velocity clamping scheme was not used in PSO with velocity adaptation.

Both standard PSO and PSO with velocity adaptation were run with two different bound handling mechanisms: Reflect-Z and Infinity. Furthermore, the performance of the new algorithm was also compared to standard PSO with Hyperbolic velocity update (without velocity clamping), as this configuration was among the best-performing algorithms in the experiments presented in Chapter 4. The threshold for the success rate was set to $\rho = 0.2$, and the initial step size of PSO with velocity adaptation was set to half of the average search space range:

$$l = \frac{\sum_{d=1}^{n} \frac{ub_d - lb_d}{2}}{n} \tag{5.4}$$

Note that, if the search space is an n-dimensional cube $\mathcal{S} = [-r, r]^n$, then $l = r$ holds. Each run terminated after 300,000 function evaluations, and each configuration was repeated 100 times. The following abbreviations are used:

- *Reflect-S:* Standard PSO with velocity clamping and Reflect-Z bound handling.
- *Infinity-S:* Standard PSO with velocity clamping and Infinity bound handling (same setting as Infinity-C in Chapter 4).
- *Reflect-A:* PSO with velocity adaptation and Reflect-Z bound handling.
- *Infinity-A:* PSO with velocity adaptation and Infinity bound handling.
- *Hyperbolic:* Standard PSO with Hyperbolic velocity update and without velocity clamping (same setting as Hyperbolic in Chapter 4).

PSO with velocity adaptation was tested and compared to a standard PSO algorithm on all 100- and 500-dimensional benchmarks. Sample means of the obtained final objective values, respective 95% confidence intervals and standard deviations are presented in Tables 5.5, B.28, and B.29. The results of the Wilcoxon rank sum tests can be found in Tables 5.6, B.26, and B.27.

Table 5.4: Setup for the experimental analysis.

PSO parameters	
Population size	49
Neighborhood topology	7×7 grid (self included, undirected)
Accelaration coefficients c_1 and c_2	1.496172
Inertia weight ω	0.72984
Craziness (mutation, turbulence)	no
Particle initialization	Uniformly at random in the search space
Velocity initialization	Half-diff initialization

On most benchmarks, the adaptive particle swarm optimizers Reflect-A and Infinity-A clearly outperformed their non-adaptive counterparts. An example is given in Figure 5.10. Out of in total 18 benchmarks per dimensionality, Reflect-A significantly outperformed Reflect-S on 14 100-dimensional and on 15 500-dimensional problems. Even more frequently, Infinity-A significantly outperformed Infinity-S (see Table 5.6). The obtained final objective values were often considerably improved (see Table 5.5). Moreover, PSO with velocity adaptation often produced significantly better results than Hyperbolic velocity update although this algorithm performed exceptionally well in the experiments presented in Chapter 4.

Summarized, the new algorithm provided significantly better results than a standard particle swarm optimizer on most benchmarks. However, on Schwefel, f9, and f10, PSO with velocity adaptation cannot compete with Reflect-S. An example convergence plot is shown in Figure 5.11. These are the same benchmarks that were only comparatively poorly solved with Hyperbolic velocity update in high dimensions. Both f9 and f10 have a huge number of local optima [SHL$^+$05]. When solving Schwefel, f9, or f10, particle velocities often rapidely decrease towards zero, diminishing the exploration capabilitiy of the PSO algorithm, which is crucial for the successful optimization of highly multi-modal problems. In such a situation, the introduction of a maximum number of velocity reduction steps might be beneficial.

PSO with velocity adaptation was designed on the basis of the theoretical results presented in Section 3.4 with the aim of reducing the effect of bound handling on particle swarm performance when solving high-dimensional problems. From Table 5.6, the following information can be extracted for the 100-dimensional benchmarks:

- 100-dimensional benchmarks: Standard PSO
  - Reflect-S significantly outperformed Infinity-S on 5 benchmarks
  - Infinity-S significantly outperformed Reflect-S on 5 benchmarks

Table 5.5: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated algorithms on various 100-dimensional benchmarks. The best objective values are presented together with the function name.

	Sphere (0)	Rosenbrock (0)	f1 (-450)
Hyperbol.	5.95e-06±1.7e-07 (8.4e-07)	152.9±9.423 (47.49)	**-450 (0)**
Reflect-S	*6.17e-06±1.9e-07 (9.4e-07)*	*201.3±11.53 (58.11)*	**-450 (0)**
Reflect-A	**1.04e-06±1.4e-08 (6.9e-08)**	**114.2±7.311 (36.85)**	**-450 (0)**
Infinity-S	6.07e-06±1.9e-07 (9.5e-07)	199.8±10.72 (54.02)	**-450 (0)**
Infinity-A	1.05e-06±1.4e-08 (6.9e-08)	123.3±7.78 (39.21)	**-450 (0)**
	Griewank (0)	Rastrigin (0)	Schwefel (≈ -41898.3)
Hyperbol.	3.57e-03±1.4e-03 (7.0e-03)	105.5±4.193 (21.13)	-24848±333.6 (1681)
Reflect-S	*4.72e-03±3.9e-03 (0.02)*	*296.6±8.581 (43.25)*	**-31015±280.2 (1412)**
Reflect-A	**6.42e-04±5.5e-04 (2.7e-03)**	93.61±3.512 (17.7)	-25448±285.2 (1437)
Infinity-S	3.13e-03±1.1e-03 (5.6e-03)	261.8±6.92 (34.88)	-23923±312.3 (1574)
Infinity-A	1.09e-03±6.7e-04 (3.4e-03)	**91.61±2.927 (14.75)**	*-22832±222.9 (1123)*
	f3 (-450)	f2 (-450)	f9 (-330)
Hyperbol.	1.21e+07±446800 (2.25e+06)	3348±123.6 (622.7)	*264.95±9.73 (49.04)*
Reflect-S	*3.57e+07±1.87e+06 (9.4e+06)*	*21047±1037 (5228)*	**-0.73678±9.63 (48.54)**
Reflect-A	8.18e+06±345800 (1.74e+06)	333.2±33.56 (169.2)	119.05±13 (65.51)
Infinity-S	1.76e+07±1.06e+06 (5.34e+06)	10696±640.9 (3230)	228.93±13.94 (70.25)
Infinity-A	**7.14e+06±279300 (1.41e+06)**	**264.6±36.15 (182.2)**	159.68±12.28 (61.9)
	f8 (-140)	f6 (390)	f5 (-310)
Hyperbol.	-118.71±6.444e-03 (0.03248)	570.7±8.938 (45.04)	27956±572.4 (2885)
Reflect-S	*-118.7±5.315e-03 (0.02679)*	584.6±9.83 (49.54)	27310±765.7 (3859)
Reflect-A	**-118.93±8.091e-03 (0.04078)**	**514.4±6.908 (34.81)**	**23575±611.2 (3080)**
Infinity-S	*-118.7±5.987e-03 (0.03017)*	*592.3±10.54 (53.1)*	*34700±896.7 (4519)*
Infinity-A	-118.92±8.069e-03 (0.04066)	515.3±7.359 (37.09)	26909±518.5 (2613)
	Ackley (0)	f10 (-330)	f11 (-460)
Hyperbol.	0.37429±0.128 (0.645)	*478.7±15.56 (78.4)*	214.65±1.531 (7.718)
Reflect-S	1.5427±0.173 (0.8721)	**4.413±10.03 (50.57)**	217.99±1.213 (6.116)
Reflect-A	**3.68e-06±2.4e-08 (1.2e-07)**	59.53±17.3 (87.2)	**184.69±1.595 (8.037)**
Infinity-S	*1.7253±0.1398 (0.7047)*	280.8±22.33 (112.6)	*223.3±1.434 (7.229)*
Infinity-A	3.70e-06±2.5e-08 (1.2e-07)	174.4±11.56 (58.27)	184.9±1.721 (8.675)
	f14 (-300)	f13 (-130)	f12 (90)
Hyperbol.	**-254.18±0.1192 (0.6007)**	-102.8±0.934 (4.707)	173200±15810 (79690)
Reflect-S	*-253.46±0.0754 (0.38)*	-68.4±3.052 (15.38)	*559510±48170 (242800)*
Reflect-A	-254.1±0.1119 (0.5641)	-112.1±0.6355 (3.203)	238170±34790 (175300)
Infinity-S	-253.63±0.08549 (0.4308)	*-65.19±2.592 (13.06)*	293280±23700 (119500)
Infinity-A	-254.02±0.1039 (0.5237)	**-112.4±0.5849 (2.948)**	**163930±16330 (82310)**

Table 5.6: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows how often A performed significantly better than B. The total number of benchmarks is 18.

100-dimensional benchmarks

	1	2	3	4	5
Hyperbolic (1)	0	10	13	1	1
Reflect-S (2)	3	0	5	3	3
Infinity-S (3)	2	5	0	0	1
Reflect-A (4)	15	14	17	0	4
Infinity-A (5)	14	13	16	3	0

500-dimensional benchmarks

	1	2	3	4	5
Hyperbolic (1)	0	14	18	6	4
Reflect-S (2)	3	0	18	3	3
Infinity-S (3)	0	0	0	0	0
Reflect-A (4)	12	15	18	0	3
Infinity-A (5)	13	15	18	4	0

⇒ In total 10 significant differences

- 100-dimensional benchmarks: PSO with velocity adaptation
  - Reflect-A significantly outperformed Infinity-A on 4 benchmarks
  - Infinity-A significantly outperformed Reflect-A on 3 benchmarks

  ⇒ In total 7 significant differences

Hence, the impact of bound handling on the final solution quality was hardly reduced by velocity adaptation when solving 100-dimensional benchmark problems. The situation is different for the 500-dimensional benchmarks: these problems were only poorly solved by Infinity-S, despite the fact that velocity clamping was used. Accordingly, Reflect-S significantly outperformed Infinity-S on all 18 benchmarks, while only 7 significant differences occured between Reflect-A and Infinity-A (see Table 5.6). This means that there is some evidence that the significance of bound handling can be reduced by velocity adaptation, but further investigation is needed. Therefore, the following algorithms were introduced in this experiment:

- *Nearest-S:* Standard PSO with velocity clamping and Nearest-Z bound handling.
- *Random-S:* Standard PSO with velocity clamping and Random-Z bound handling.
- *Nearest-A:* PSO with velocity adaptation and Nearest-Z bound handling.
- *Random-A:* PSO with velocity adaptation and Random-Z bound handling.

The results of the Wilcoxon rank sum test are summarized in Table 5.7 for the 500-dimensional problems. They show that bound handling becomes less important for a PSO algorithm if velocity adaptation is used: 90 versus 44 significant performance

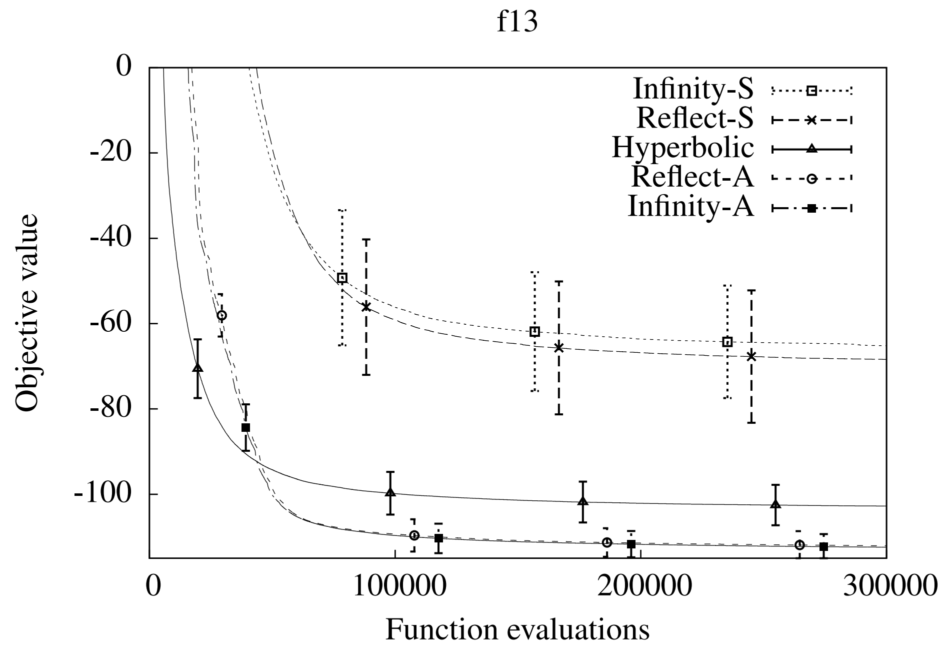


Figure 5.10: On many 100- and 500-dimensional functions, the adaptive PSO algorithms significantly outperformed their non-adaptive counterparts, as exemplarily shown for the 100-dimensional f13 benchmark. PSO with velocity adapation often also significantly outperformed PSO with Hyperbolic velocity update. In the plot, average values and standard deviations (vertical bars) are shown.

differences. However, the final solution quality still strongly depended on the bound handling method in some cases (see Table B.30). In particular, Random-A was often outperformed by the other adaptive algorithms in higher dimensions. Using this bound handling method can lead to a very exploratory, rather random swarm behavior on high-dimensional problems, as already discussed in Section 4.4.2. Moreover, the particle swarm might have difficulties to approach boundary regions (see discussion in Section 4.4.2 for more details). The experimental results suggest that these effects were not completely eliminated by velocity adaptation. It can be concluded that PSO with velocity adaptation is less sensitive to the choice of the bound handling method than a standard particle swarm optimizer. However, invariance was not achieved. Experiment 5 will show that a smaller initial step size can decrease the importance of bound handling, but can also deteriorate the algorithm's performance.

Summarized, PSO with velocity adaptation yielded superior results than a standard PSO on most of the investigated 100- and 500-dimensional benchmarks. The impact of bound handling on particle swarm performance was reduced.

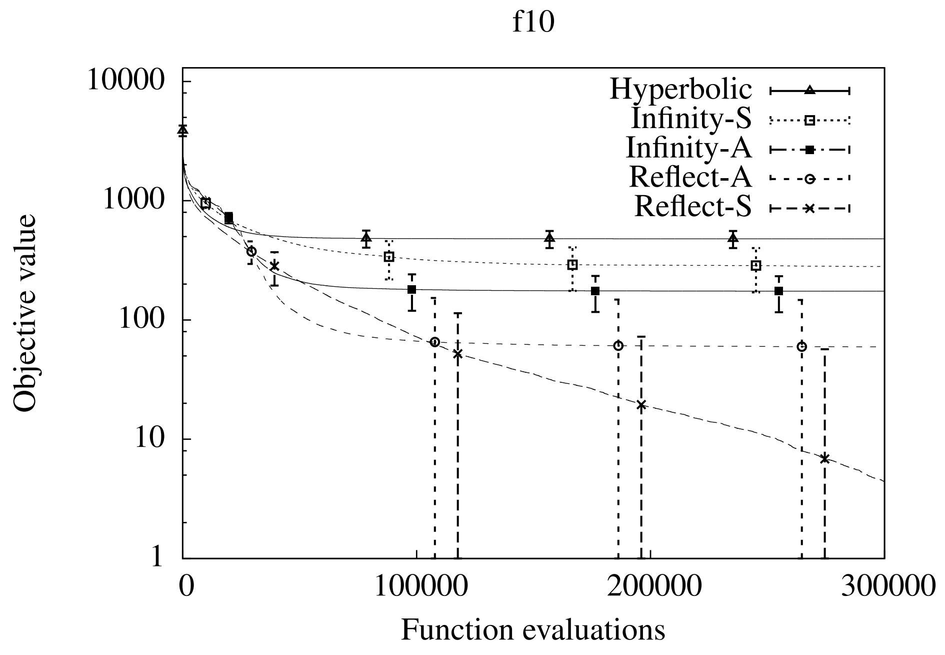


Figure 5.11: Reflect-S produced significantly better results than its adaptive counterpart on the 100- and 500-dimensional Schwefel, f9, and f10. Only once, Infinity-S significantly outperformed Infinity-A (100-dimensional Schwefel benchmark). The plot shows average results and standard deviations (vertical bars) for the 100-dimensional benchmark f10. Note that a logarithmic scale is used.

Table 5.7: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows how often A performed significantly better than B (500-dimensional benchmarks).
Left: Detailed results. Right: Sum of significant performance differences in the corresponding areas.

	1	2	3	4	5	6	7	8
Reflect-S (1)	0	18	9	9	3	3	3	8
Infinity-S (2)	0	0	0	0	0	0	0	0
Nearest-S (3)	2	18	0	9	3	3	3	8
Random-S (4)	4	18	3	0	3	3	3	7
Reflect-A (5)	15	18	15	15	0	3	2	9
Infinity-A (6)	15	18	15	15	4	0	4	10
Nearest-A (7)	15	18	15	15	0	2	0	10
Random-A (8)	10	18	10	10	0	0	0	0

	Standard	Adapative
Standard	90	
Adaptive		44

Experiment 2: Threshold ρ

In accordance to Rechenberg's $1/5$-rule [Rec73], the threshold for the success rate was set to $\rho = 0.2$ in the first experiment. In this experiment, additional settings are analyzed: $\rho = 0.01, 0.1, 0.5, 0.8$.

Sample means of the obtained final objective values per setting, corresponding 95% confidence intervals, standard deviations, and the results of the Wilcoxon rank sum tests are presented in the appendix in Tables B.33, B.34, B.32, and B.31. Selected results are shown in Table 5.8 for convenience. In Figure 5.12, the convergence plots of two representative runs are shown.

Velocity adaptation has strong impact on the behavior of a particle swarm. Each $n = 100$ iterations, i.e., after each $100 \cdot 49 = 4,900$ function evaluations, adaptation takes place. These periodic adaptation steps are often clearly visible in the convergence plots, as shown for f6 in Figure 5.13. In the top figure, the complete convergence plot is shown, while the bottom figure only shows the initial steps of the PSO algorithm. The first adaptation took place after 4,900 function evaluations. At that point, particle velocities were often doubled if $\rho \in \{0.01, 0.1, 0.2\}$, and halved if $\rho = 0.5$ or $\rho = 0.8$, when solving the benchmark problem f6. The differences in the swarm behavior are visible as a split of the convergence graphs in Figure 5.13 (bottom). The next splits occured after $200 \cdot 49 = 9,800$ and $300 \cdot 49 = 14,700$ objective function evaluations.

When choosing a high threshold ρ, particle velocities often rapidly approach zero, which prevents thorough search space exploration, and which can lead to premature convergence. Accordingly, setting ρ to 0.5 or 0.8 resulted in poor average objective values on most benchmarks (see Tables 5.8, B.33, and B.34).

The performance of Reflect-A-0.1 (i.e., $\rho = 0.1$ and Reflect-Z bound handling) and Reflect-A-0.2 was quite similar with slight advantage for Reflect-A-0.1, considering average objective values and the result of the Wilcoxon rank sum tests. The behavior of choosing $\rho = 0.01$ differs from the other settings: The convergence speed was often too slow to obtain satisfactory results during $300,000$ function evaluations (which already is quite much). An example is given in Figure 5.12 (top). A similar behavior was observed on approximately 8 other functions (Sphere, Rosenbrock, Ackley, Griewank, f1, f3, f6, f12). When using such small values for ρ, particle velocities are comparatively high. In Table 5.9, the average distance a particle moved per iteration is shown for a representative set of benchmark functions. The smaller ρ, the higher were the particle velocities. High particle velocities can lead to a rather random behavior of the whole swarm and considerably slow down the convergence speed. On the other hand, the same behavior is advantageous for other problems, see Figure 5.12 (bottom): In these cases, the use of high thresholds led to premature convergence, as velocities are decreased further and further if the swarm is not successful. However, the swarms with $\rho = 0.01$ kept exploring and finally found solutions of much better quality than the other variants on approximately 7 of the investigated 18 benchmark

Table 5.8: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies on selected 100-dimensional benchmarks. The best objective values are presented together with the function name.

	f2 (-450)	f6 (390)	f9 (-330)
Reflect-A-0.01	*22994±575.93 (2902.5)*	*798.4±43.341 (218.43)*	**-78.783±9.1213 (45.97)**
Reflect-A-0.1	1685.9±96.253 (485.09)	**500.15±5.1833 (26.123)**	90.245±9.0306 (45.51)
Reflect-A-0.2	**333.15±33.564 (169.15)**	514.39±6.9077 (34.813)	119.05±13 (65.52)
Reflect-A-0.5	2102±282.69 (1424.7)	554.58±9.7745 (49.261)	182.24±10.041 (50.60)
Reflect-A-0.8	14435±841.11 (4239)	581.78±10.02 (50.498)	*186.59±8.6135 (43.41)*

Table 5.9: Average distance a particle moved per iteration and corresponding 95% confidence intervals for a representative set of 100-dimensional benchmarks.

	f2	f6	f9	f11
Reflect-A-0.01	77.2±0.55	63.5±1.90	4.74±0.0939	0.154±0.00262
Reflect-A-0.1	17.5±0.094	18.0±0.00208	0.434±8.61e-04	0.0196±0.000827
Reflect-A-0.2	11.8±0.030	10.2±0.0999	0.331±0.0116	0.0164±9.70e-07
Reflect-A-0.5	4.39±0.0089	4.26±0.185	0.164±1.98e-05	0.0164±8.38e-07
Reflect-A-0.8	3.65±0.0038	3.29±0.000396	0.164±6.04e-06	0.016±8.37e-07

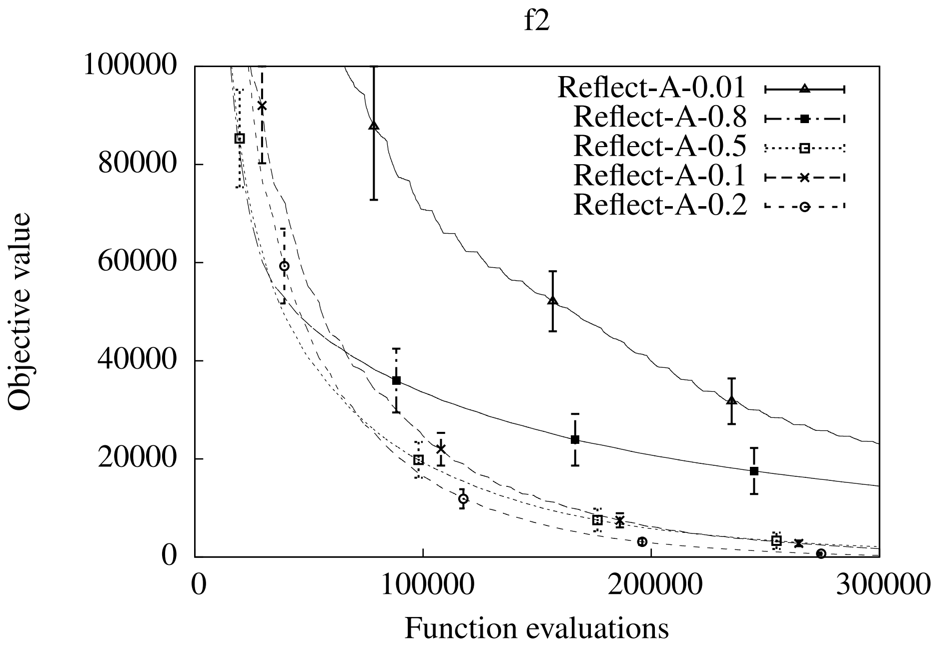
functions (Schwefel, f5, f8, f9, f10, f11, f13).

Summarized, the use of very small or very high values for the threshold ρ cannot be recommended. Using a very low threshold sometimes led to exceptionally good results, but also often deteriorated particle swarm performance. However, selecting $\rho = 0.1$ or $\rho = 0.2$ delivered solutions of good quality for most investigated problems.

Experiment 3: Scaling Strategy

As already mentioned, and in contrast to previously proposed (static and dynamic) velocity clamping methods, PSO with velocity adaptation involves a so-called *up- and down-scaling* mechanism. This means that not only too large velocities are cut, but also too small velocities are increased. This way, the particle swarm's exploration and exploitation behavior can more directly be controlled. In this experiment, the proposed *up- and down-scaling* approach is compared to the use of an adaptive maximum particle velocity, which can be described as follows: Particle velocities that exceed the given adaptive velocity limit l_v are decreased, small velocities are, however, not scaled up. As effectively a maximum step size is used, this setting is denoted as *maximum velocity* or, abbreviated, as *maxvel* in the following tables.

The results of the one-sided Wilcoxon rank sum test are shown in Tables 5.10



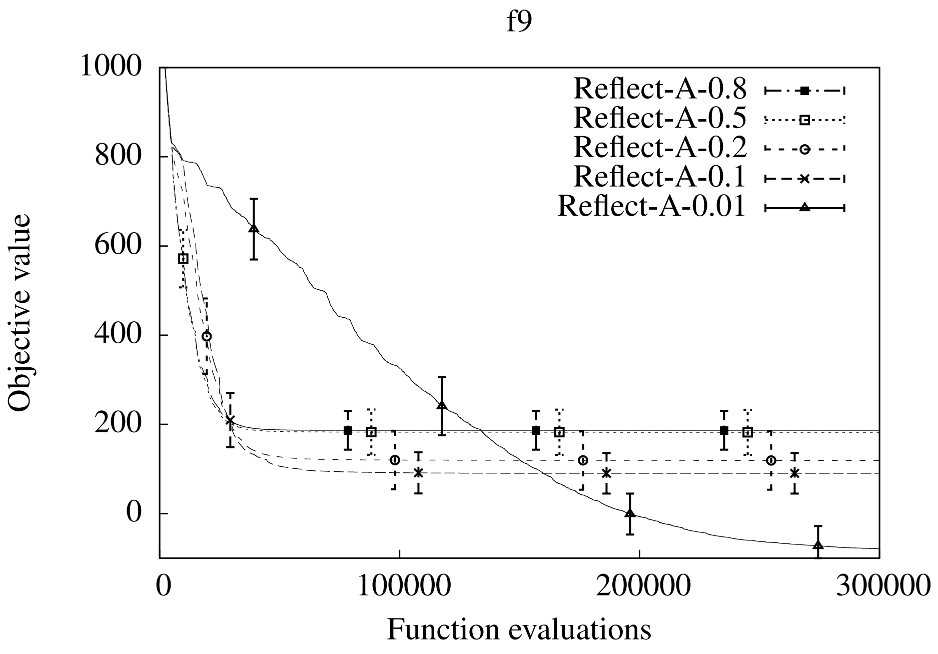
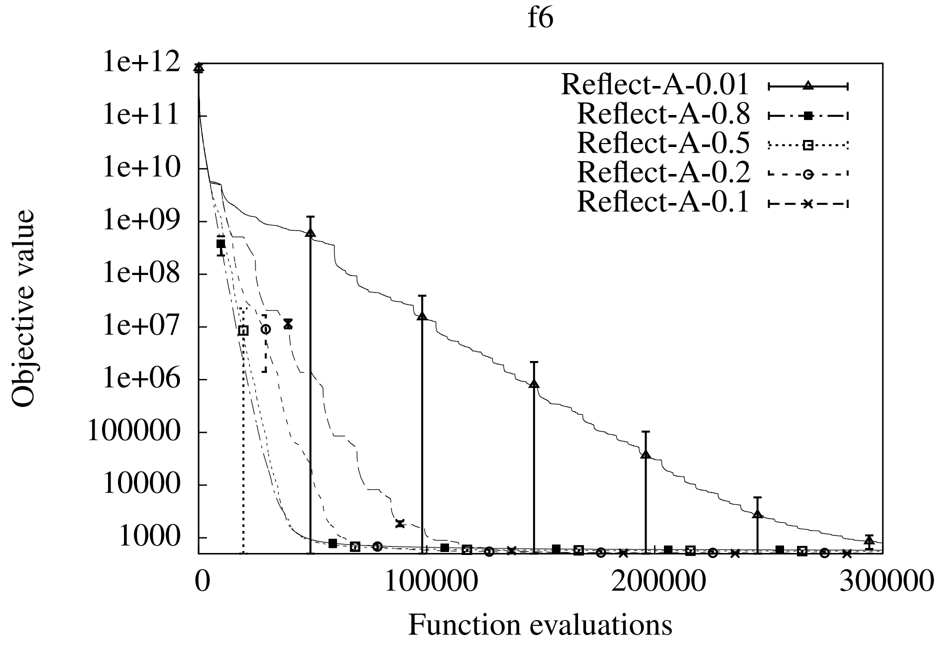


Figure 5.12: Representative runs (top: f2, bottom: f9) of the adaptive PSO algorithm with different settings of ρ (Experiment 2). In the plot, average objective values are shown. Vertical bars depict the standard deviation.



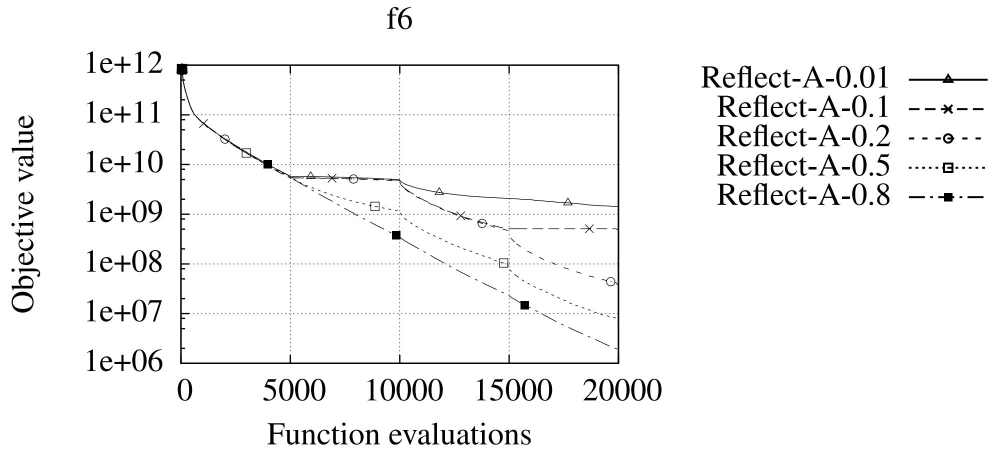


Figure 5.13: Velocity adaptation has strong impact on the algorithmic behavior. The adaptation steps are often clearly visible in the convergence plots, as exemplarily shown for f6. Top: Convergence plot for 300,000 function evalutions. Bottom: Initial steps of the PSO algorithm when solving f6 (sample means).

Table 5.10: Summary of the most relevant information obtained by the Wilcoxon rank sum tests for Experiment 3. Detailed results are presented in Table B.35. The table is read as follows.
Example: The first row shows that on 13 benchmarks, Reflect-A (which used the proposed up- and down-scaling mechanism) significantly outperformed Reflect-A-maxvel, while Reflect-A-maxvel never significantly outperformed Reflect-A.

	up- and down-scaling : *maximum velocity*
Reflect	13 : 0
Infinity	13 : 0

and B.35. Using a maximum step size instead of the proposed up- and down-scaling mechanism led to significantly worse final objective values on most tested benchmark problems. On the other hand, Reflect-A-maxvel and Infinity-A-maxvel never significantly outperformed their up- and down-scaling counterparts. The performance difference is also clearly visible when considering the obtained average objective values for both settings (see Table B.36). Figure 5.14 shows the convergence plot for f10 as an example.

Hence, the proposed up- and down-scaling mechanism is necessary to control the search strategy of the particle swarm. It is therefore an important feature of PSO with velocity adaptation.

Experiment 4: Personal vs. Global Step Size

PSO with velocity adaptation as presented in Algorithm 5.1 uses a global step size l_v, which is adapted to the optimization process by considering the success of the entire particle swarm. Each particle scales its velocity vector to the same length l_v. However, it is also possible that each particle i has its *personal* (or *individual*) step size $l_{i,v}$ to which its velocity is scaled, and which is adapted according to the particle's personal success. If a particle's personal success rate exceeds the threshold ρ, its step size is doubled, otherwise it is halved. Again, adaptation takes place every n iterations. In the following tables, this approach is abbreviated as *ind* (for *individual*).

This alternative setting was tested by using the given benchmark set. The results of the Wilcoxon rank sum tests, which are given in Table B.37 and summarized in Table 5.11, are inconclusive: there is no clear winner. However, when considering the obtained final objective values, the use of a global step size can be slightly preferred. The complete results are given in Table B.38, while selected results are presented in Table 5.12. For most benchmark functions, solutions of similar quality were achieved by both algorithms. As an example, consider the results obtained for benchmark f2. However, there are two functions for which the use of a global step size considerably

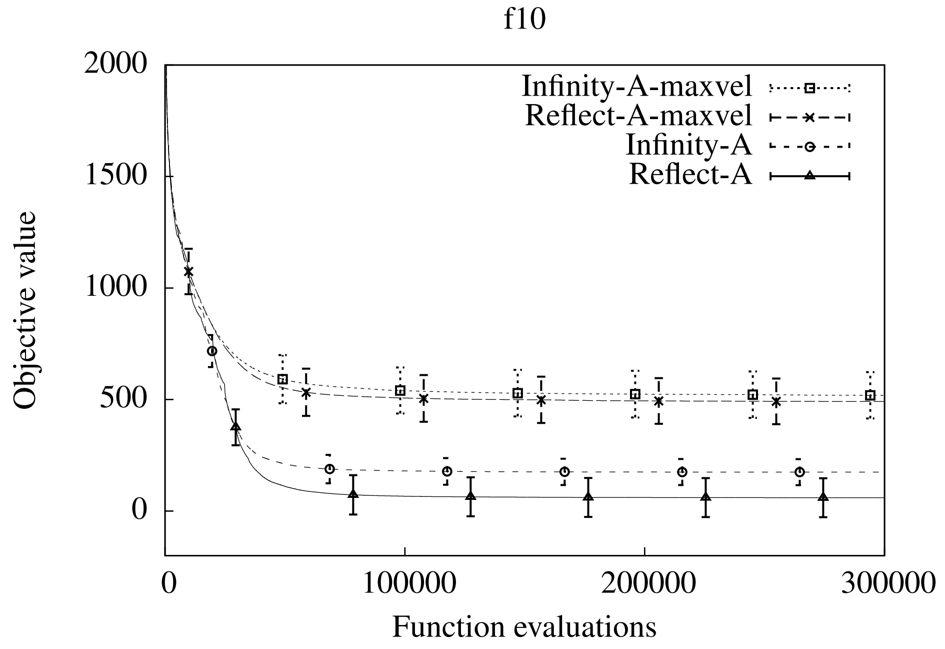


Figure 5.14: Convergence plot of PSO algorithms with different scaling strategies (Experiment 3) for benchmark function f10. On most benchmarks, the proposed up- and down-scaling mechanism led to significantly better results than the usage of an adaptive maximum value for the particles' velocities. The latter setting is denoted as *maxvel*.

Table 5.11: Summary of the most relevant information obtained by the Wilcoxon rank sum tests for Experiment 4. Detailed results are presented in Table B.37. The table is read as described in the caption of Table 5.10 on page 151.

	global step size : *individual step size*
Reflect	4 : 4
Infinity	2 : 3

Table 5.12: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated strategies (Experiment 4) on selected 100-dimensional benchmarks. The best objective values are presented together with the function name.

	f2 (-450)	f9 (-330)	f10 (-330)
Reflect-A	333.2±33.56 (169.2)	**119.1±13 (65.52)**	**59.53±17.30 (87.20)**
Reflect-A-ind	*364.8±36.79 (185.4)*	166.1±9.531 (48.033)	176.9±19.79 (99.72)
Infinity-A	**264.6±36.15 (182.2)**	159.7±12.28 (61.898)	174.4±11.56 (58.28)
Infinity-A-ind	279.2±28.79 (145.1)	*196.1±9.598 (48.37)*	*304.85±19.51 (98.31)*

improved the final solution quality: f9 and f10. Therefore, the use of a global step size is recommended for PSO with velocity adaptation, although this setting mostly did not strongly affect the algorithmic behavior.

Experiment 5: Initial Step Size

In the previous experiments, the particles' step sizes were initialized according to Equation (5.4). If the search space $\mathcal{S}$ is an n-dimensional cube $\mathcal{S} = [-r, r]^n$, $l = r$ holds. This roughly corresponds to a order of $r/\sqrt{n}$ per problem dimension. In the theoretical analysis presented in Section 3.4, it was shown that the probability that a particle becomes infeasible at time step t is $1 - \left(1 - \frac{1}{4s}\right)^n$, assuming that the particle velocities are drawn uniformly at random in $[-\frac{r}{s}, \frac{r}{s}]$, $s \geq 1$. For $s = \sqrt{n}$, this probability approaches 1 with increasing n.

In this experiment, the initial step size was set to $l = r/\sqrt{n}$, or, more generally, to

$$l = \frac{\sum_{d=1}^{n} \frac{ub_d - lb_d}{2}}{\sqrt{n}} \tag{5.5}$$

if the search space is $\mathcal{S} = [lb_1, ub_1] \times \ldots \times [lb_n, ub_n]$. This setting corresponds to a order of r/n per dimension, and should result in less particles to become infeasible, since $1 - \left(1 - \frac{1}{4n}\right)^n$ approaches the constant $1 - e^{-\frac{1}{4}} < 1$ for large n. Both initializa-

Table 5.13: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows how often A performed significantly better than B on the 100-dimensional benchmarks. The total number of benchmark functions is 18.

	1	2	3	4
Reflect-A (1)	0	4	10	8
Infinity-A (2)	3	0	10	10
Reflect-A-init2 (3)	3	3	0	1
Infinity-A-init2 (4)	3	3	2	0

tion strategies were analyzed on the given benchmark set. The method presented in Equation (5.5) is denoted as *init2*.

A comparison of the obtained average objective values, which are presented in Table B.40, suggests that setting the initial step size to $l = r$ is to be preferred over $l = r/\sqrt{n}$. On most benchmarks, significantly better results were achieved if the initial step size was set to the larger value $l = r$. As an example, the convergence plot for the Schwefel function is depicted in Figure 5.15. Especially for Schwefel, f9 and f10, great performance losses were recognized if the initial step size was set to $l = r/\sqrt{n}$.

However, as indicated by the theoretical study presented in Section 3.4, choosing $l = r/\sqrt{n}$ (which corresponds to the order r/n per dimension) resulted in a PSO algorithm whose performance is more independent to the choice of the bound handling method: Table 5.13 shows that there were less significant performance differences between *Reflect-init2* and *Infinity-init2* than between *Reflect-A* and *Infinity-A*.

Considering Table 5.13 (more details are given in Table B.39), it is clear that *Reflect-A* and *Infinity-A* more often significantly outperformed the corresponding *init2* algorithms than vice versa. In the three cases (Sphere, Ackley, and Griewank) in which the *init2* variants produced significantly better solutions than the adaptive PSO with $l = r$, both settings provided a satisfactory average solution quality very close to the global optimum. Hence, the experimental results suggest to set the initial step size rather to $l = r$ than to $l = r/\sqrt{n}$, although more particles are expected to leave the feasible space. The higher the initial step size, the more exploration takes place at the beginning of the optimization, which is mostly a desired feature of a search heuristic.

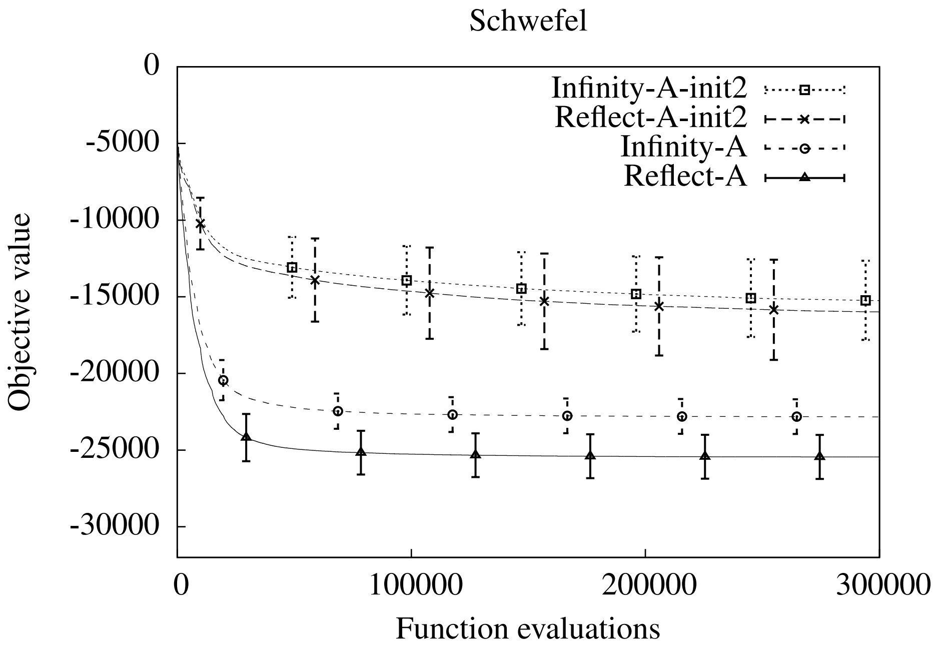


Figure 5.15: Convergence plot of PSO algorithms with different initial step sizes (Experiment 5) for the Schwefel benchmark.

6. Application of PSO to Relative Positioning

The previous chapters were dedicated to the theoretical analysis of PSO for box-constrained problems, thorough experimentation on artifical benchmarks, and the derivation of two adaptive PSO algorithms. In this chapter, these studies are complemented by a real-world PSO application, the solving of the so-called relative positioning problem. The results presented in this chapter were mostly published in [SWHP07, GWHK09]. The investigated optimization task is a six-dimensional box-constrained problem. The impact of bound-handling on this rather low-dimensional optimization problem is analyzed. Furthermore, both adaptive algorithms presented in the previous chapter were applied on relative positioning, and compared with standard particle swarm optimization.

6.1 Background

The optimization problem investigated in this chapter is settled in the field of tolerance analysis in mechanical engineering. Figure 6.1 shows a flowchart of the tolerance analysis process as described by Pierce and Rosen [PR97] and Wittmann et al. [WSP07]. The approach can for instance be used for the development of car components or machine parts. The first step is the design of the ideal assembly, also denoted as *model*, with computer-aided design (CAD) software or other means. As an example, the model of a crossbeam and its surrounding parts (also denoted as its environment), a car underbody, is shown in Figure 6.2[1]. However, parts with ideal geometry cannot be produced. Instead, small deviations appear during the manufacturing process. If the deviations are too large, functional or aesthetic requirements might be violated. Therefore, allowed ranges for the deviations, so-called *tolerances*, are specified by the product designer.

The definition of appropriate tolerances is an important step in product development. The tighter the defined tolerances, the better is the quality of the product, but the higher are usually the manufacturing costs. In order to find a good trade-off, the effects of the defined tolerances on product functionality and appearence are ana-

[1]The author likes to thank Stefan Wittmann from the Department of Mechanical Engineering, University of Erlangen-Nuremberg, for Figures 6.2, 6.3, and 6.4.

lyzed. The results of the analysis are then used to redefine the specified tolerance ranges if necessary.

In the given process, the impact of the defined tolerances is analyzed by using statistical approaches [NT95] and/or visualization methods [WSP07]. For the analysis, sample manufactured parts (part variants) are needed, which can for instance be obtained by a simulation of the manufacturing process or by measurements conducted on real, manufactured parts. In order to analyze and visualize the impact of the defined tolerance ranges, the part variants have to be positioned in the environment. This step is called *relative positioning* [Tur90]. Figure 6.3 shows the model of an ideal crossbeam (bright color) and a simulated, positioned crossbeam (dark color) in their environment. The part variants have to be positioned such that several criteria are fulfilled, for instance, non-interference with the environment and proximity to other parts. The relative positioning step can therefore be formulated as an optimization task [Tur90, PR97, SWHP07]. After the relative positioning step, the statistical and visual analysis takes place. If necessary, tolerances are redefined afterwards and the process is repeated (see Figure 6.1).

6.2 Problem Description and Optimization Framework

The optimization problem considered in this section is called *relative positioning*. The task is to position a simulated or measured part in its environment such that several criteria, for instance, closeness to the environment and non-interference, are fulfilled. If a single part is to be positioned in a three-dimensional space, the search space has six dimensions: three translational and three rotational ones.

Turner [Tur90] and Sodhi and Turner [ST94] used a mathematical programming approach for the relative positioning of parts in the context of tolerance analysis. For different assemblies they propose different objective functions, for instance, the minimization of the maximum distance between two neighboring parts. Non-interference of the participating parts is formulated as a set of constraints. Under certain circumstances, the task can be formulated as a linear programming problem and solved efficiently. Otherwise, it can be approached by metaheuristic optimization algorithms. Pierce and Rosen [PR97] used simulated annealing for this purpose. Alternatively, evolution strategies or particle swarm optimization can be utilized [SWHP07, GWHK09].

In various medical imaging and computer vision applications, images must be aligned such that their mutual distance is minimized. This task, known as *registration*, is similar to relative positioning (see [ZF03] for a survey on image registration). However, general registration methods, such as the widely-used *ICP algorithm* of Besl and McKay [BM92] are not applicable in the given scenario due the presence

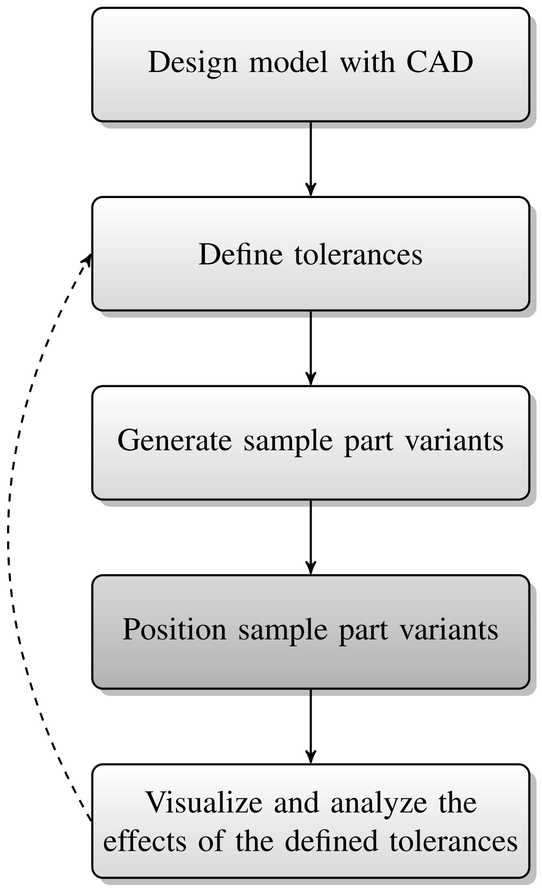

Design model with CAD
Define tolerances
Generate sample part variants
Position sample part variants
Visualize and analyze the effects of the defined tolerances

Figure 6.1: Flowchart of the tolerance analysis process as described by Pierce and Rosen [PR97] and Wittmann et al. [WSP07]. PSO was used for the highlighted step in this flowchart, for the so-called *relative positioning* of simulated part variants.

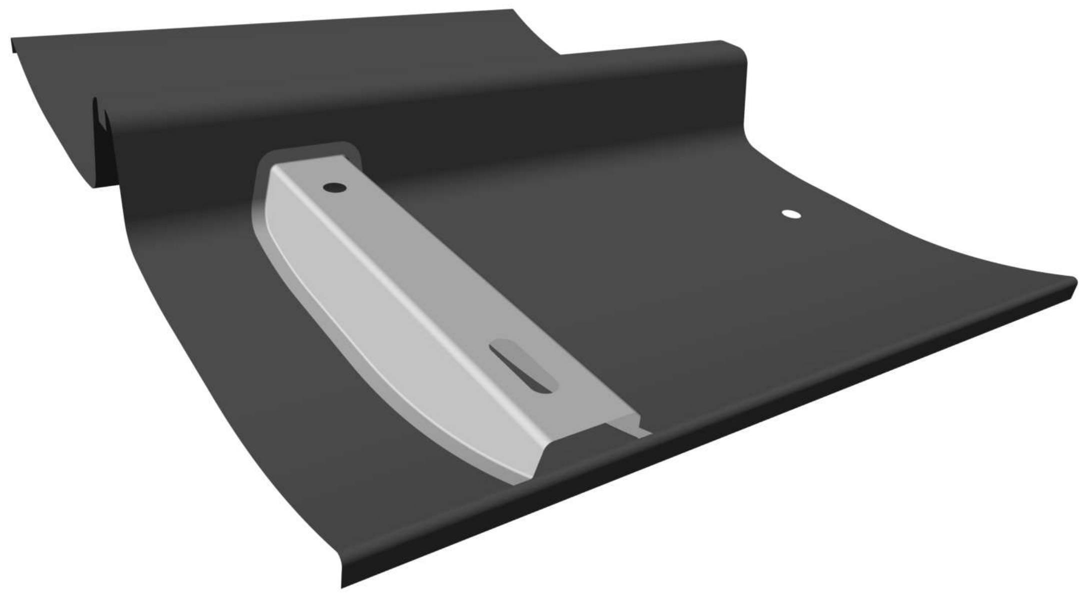


Figure 6.2: Model of crossbeam and its environment, a car underbody.

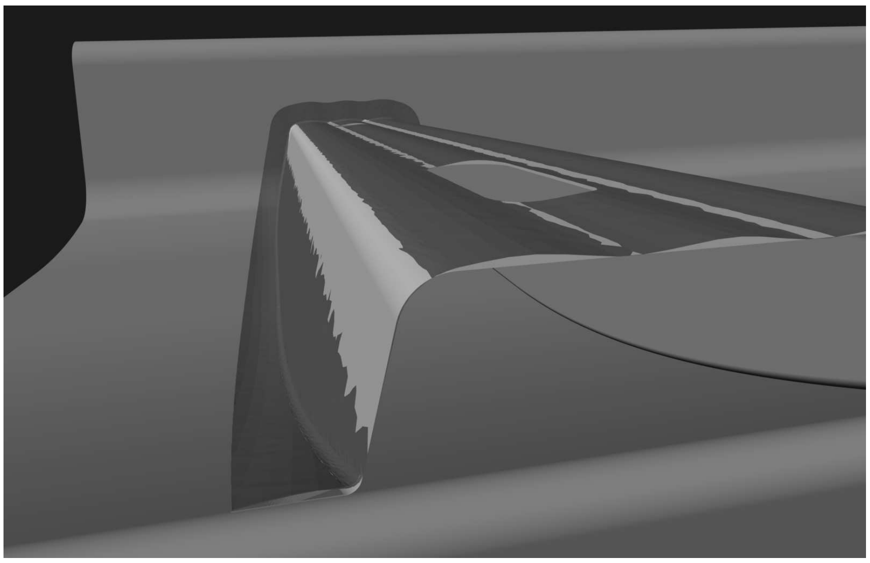


Figure 6.3: Ideal (bright color) and non-ideal, positioned (dark color) crossbeam and its environment.

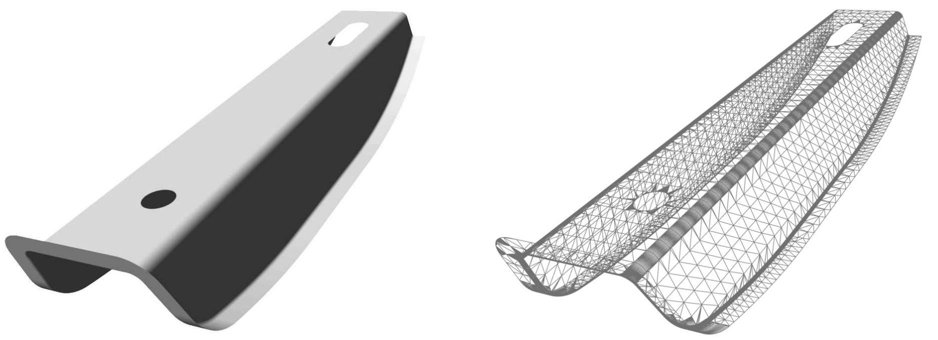


Figure 6.4: Model of ideal crossbeam, and corresponding triangle mesh.

of the non-interference constraint. Moreover, by using metaheuristic optimization approaches for the relative positioning problem instead of a (modified) registration technique, the optimization framework is by far more flexible: Additional problem-specific objectives, such as gravity or tolerance definitions that affect the relative positions of the parts, can easily be included [SWHP07].

In the following case study, a crossbeam is to be positioned relative to its environment, a car underbody. The ideal situation without manufacturing deviations is depicted Figure 6.2. Both crossbeam and car underbody are represented as fine-grained triangle meshes as shown in Figure 6.4. The fine-granularity of the triangle mesh is important to be able to introduce small deviations in the model [SWM09]. After modeling, triangularization, and the definition of tolerances, the first step is to generate non-ideal parts, for instance, by a simulation of the manufacturing process. This task was done by using the methodology of Stoll [Sto06] (see also [SWM09]): In order to generate a non-ideal part, the vertices of the triangle mesh are slightly modified such that the shape of the resulting part still complies with the tolerance definition.

Subsequently, the non-ideal part is to be positioned in its environment, in order to visually and statistically investigate the impact of the introduced deviations on product functionality and aesthetics. The following objectives were defined [SWHP07]:

- *Non-interference:* The part must not interfere with its environment. This objective is represented as a binary function $g(\vec{x})$:

$$g(\vec{x}) = \begin{cases} 1 & \text{if the non-ideal part interferes with its environment} \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

The input parameter $\vec{x}$ is a six-dimensional vector (if a single part is to be positioned in a three-dimensional space) defining the translations and rotations of the non-ideal part.

- *Distance minimization:* The part should be positioned such that its final position is as close as possible to the ideal part position. This means that the distance between non-ideal and ideal part is to be minimized. The assembly is represented as a fine-grained triangle mesh.
  - *Hausdorff distance:* Let A and B be two finite sets of points. The *directed Hausdorff distance* from set A to set B is given by (see, e.g., [HKR93]):

    $$d_h(A,B) = \max_{a \in A} \min_{b \in B} \{d(a,b)\}$$

    where $d(a,b)$ is the distance of points a and b using any appropriate distance measure (e.g., Euclidean distance).

    The *Hausdorff distance* of two sets A and B is then defined as (see, e.g., [HKR93, EM97]):

    $$d_H(A,B) = max\{d_h(A,B), d_h(B,A)\} \tag{6.2}$$

    An example is given in Figure 6.5.

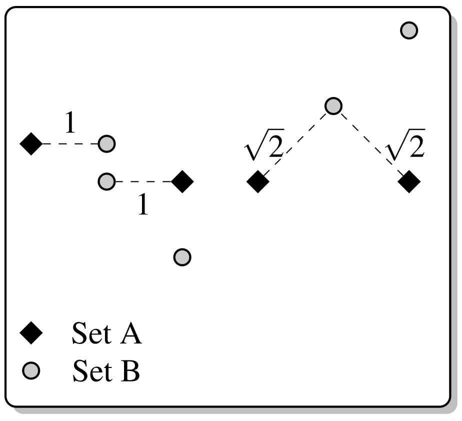
    In the relative positioning framework, a slightly modified definition of the Hausdorff distance was used, in order to better take the geometry of the participating parts into account. Instead of using the raw node sets of the involved triangle meshes, point-to-surface distances are computed. Let $A = \{a_1, a_2, \ldots, a_{|A|}\}$ and $B = \{b_1, b_2, \ldots, b_{|B|}\}$ be the node sets of the non-ideal and ideal part's triangle mesh, respectively. Furthermore, for each a_i, let b'_i be the closest point on the ideal part's surface, by using an appropriate distance measure, e.g., the Euclidean norm, and let B_A be the set $B_A = \{b'_1, b'_2, \ldots, b'_{|A|}\}$. Analogously, the set A_B is computed from A and B. The distance of non-ideal and ideal part is then defined as

    $$d'_H(A,B) = max\{d_h(A,B_A), d_h(B,A_B)\} \ . \tag{6.3}$$

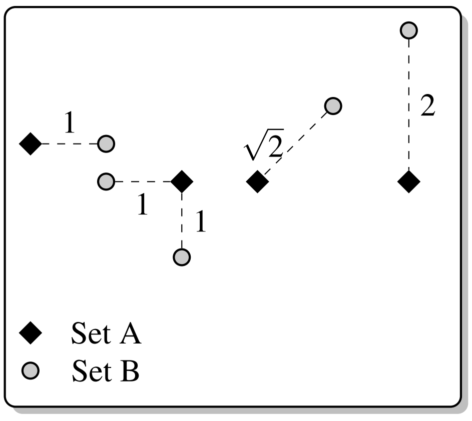
  - *Summed square distance:* The *summed square distance* is similar to the *mean square distance* used in the ICP algorithm [BM92]. It is defined as follows: For each node of the non-ideal part's triangle mesh, the minimum distance to the surface of the ideal part is computed. These distances are then squared and summed up. More formally, let A and B be the node sets of the non-ideal and ideal part's triangle mesh, respectively, and let B_A be defined as above. The summed square distance of non-ideal and ideal part computes to

    $$ssd(A,B_A) = \sum_{i=1}^{|A|} d(a_i, b'_i)^2 \tag{6.4}$$

    where $d(a_i, b'_i)$ is the distance of points a_i and b'_i.



(a)



(b)

Figure 6.5: For the computation of the distance between two point sets A and B, the *Hausdorff distance (directed or undirected)* can be used.
(a) For each element of set A, the minimum distance to an element of set B is computed (shown as dashed edges). The maximum of these distances is the directed Hausdorff distance from set A to set B: $d_h(A,B) = \max_{a\in A} \min_{b\in B} d(a,b) = \sqrt{2}$. In this example, $d(a,b)$ denotes the Euclidean distance between the two points a and b.
(b) $d_h(B,A) = \max_{b\in B} \min_{a\in A} d(b,a) = 2$
The *(undirected) Hausdorff distance* is then given by the maximum of the two directed Hausdorff distances: $d_H(A,B) = max\{d_h(A,B), d_h(B,A)\} = 2$.

For large assemblies with very fine-grained triangle meshes, the computation of the summed square distance is computationally expensive. The evaluation can be sped up by *sampling*, which means that only a certain fraction $A_{Sample} \subseteq A$ is used for the computation of the summed square distance [GWHK09]. The *estimated summed square distance* is given by

$$essd(A, B_A) = \frac{|A|}{|A_{Sample}|} \sum_{a_i \in A_{Sample}} d(a_i, b_i')^2 \ . \tag{6.5}$$

where B_A is defined as above, and, for each $a_i \in A_{Sample}$, b_i' is the closest point on the ideal part's surface. The factor $|A|/|A_{Sample}|$ was added to allow a better comparison of *ssd* and *essd*. The correlation of *essd* and *ssd* is investigated in Section 6.3 for the assembly depicted in Figures 6.2 and 6.3.

The presented approach for the relative positioning problem is very flexible due to the use of metaheuristic optimization algorithms. Additional objectives can be integrated easily. Let $f_1(\vec{x}), \ldots, f_k(\vec{x})$ be the objective functions, each to be minimized[2]. The optimization goal is to minimize the weighted sum $f(\vec{x})$ of these objectives:

$$f(\vec{x}) = \sum_{i=1}^{k} w_i \cdot f_i(\vec{x})$$

where $w_1, \ldots, w_k$ are user-defined weights. Hence, the optimization problem with multiple objective functions is transformed into a single-objective one by using a weighted sum approach. Optimization algorithms that approximate the Pareto front and return a set of solutions are not applicable in this context as the major analyses performed on the basis of this framework involve Monte Carlo simulations. This means that hundreds or thousands of parts have to be positioned for a single analysis. The weighted sum approach allows a fully automated positioning process without the need of a human decision maker once the weights are defined. The relative positioning framework is depicted in Figure 6.6.

When positioning large assemblies that are represented by fine-grained triangle meshes, the computation of certain objective functions is very time-consuming. On the other hand, the maximum translation and rotation that has to be applied to the non-ideal part in order align it with the ideal part can often be roughly estimated based on the simulation process that was used to obtain the part variants. In order to speed up the relative positioning process, these bounds were introduced as box constraints in the optimization problem.

[2]Note that $\vec{x}$ is a six-dimensional vector, which describes the translations and rotations of the non-ideal part, as mentioned at the beginning of this section. If, for instance, one of the objectives is the summed square distance $ssd(A, B_A)$, the sets A and B_A have to be computed by using this vector. Similar transformations might be necessary for other objectives.

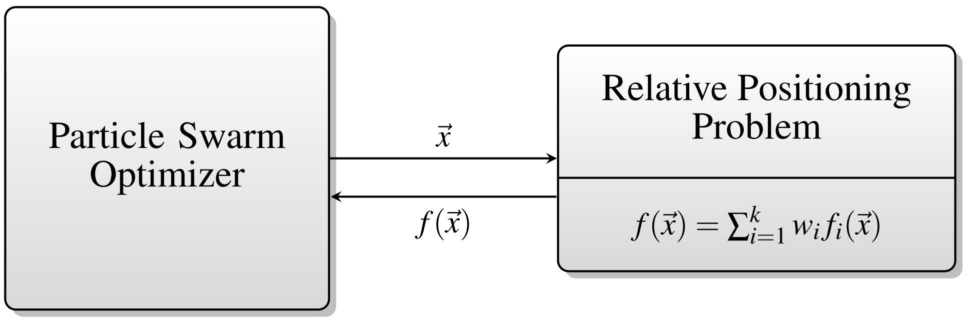

Particle Swarm Optimizer
$\vec{x}$
$f(\vec{x})$
Relative Positioning Problem
$f(\vec{x}) = \sum_{i=1}^{k} w_i f_i(\vec{x})$

Figure 6.6: Interaction of the particle swarm optimizer and the relative positioning problem. The PSO algorithm determines sample search space positions $\vec{x}$ according to its search strategy, which are evaluated by the relative positioning problem. The result $f(\vec{x})$ influences the subsequent search behavior.

6.3 Experimental Results

In this section, the presentation of particle swarm optimization for box-constrained relative positioning is concluded by experimental investigations. Two issues were addressed:

- The goal of the first experiment is to analyze the correlation of the *summed square distance (ssd)* and the *estimated summed square distance (essd)*.
- The goal of the second experiment is to compare the performance of adaptive particle swarm optimization as introduced in Chapter 5 and standard particle swarm optimization. Additionally, different strategies to cope with the box constraints were examined.

Setup

For the experimental analysis, the assembly shown in Figure 6.2 was used. The following parts are involved:

- *Environment:* The environment is a car underbody (dark color in Figure 6.2), which is described by 50,037 triangles.
- *Ideal crossbeam:* The ideal crossbeam (bright color in Figure 6.2) is represented by a triangle mesh with 14,440 triangles.
- *Simulated, non-ideal crossbeam:* The non-ideal part was obtained by introducing sinusoidal deviations into the model of the ideal crossbeam according to the method of Stoll [Sto06, SWM09]. The resulting non-ideal part is described by 49,030 triangles. The high number of triangles was necessary to represent the curved surface.

The particle swarm optimizer was instantiated with the same parameter set used in the previous experiments: A population of 49 particles was connected via a 7×7 grid topology, and initialized uniformly at random in the bounded search space. The parameters of the PSO equations were set to $c_1 = c_2 = 1.496172$ and $\omega = 0.72984$. The bound handling strategy was varied in the second experiment, but set to *Reflect-Z* in the first experiment due to its good performance in the previous experimental study. Velocities were initialized according to half-diff initialization, and neither velocity clamping nor turbulence was used.

Correlation Analysis

The correlation of *ssd* and *essd* was investigated by performing ten runs of a particle swarm optimizer using *essd* with $|A_{Sample}| = \lfloor 0.0025 \cdot |A| \rfloor = 122$ on the assembly depicted in Figures 6.2 and 6.3 and described above. The results of the correlation analysis of course only apply to the investigated assembly. However, the method can easily be used for other assemblies as well. As one of the main application areas of the relative positioning framework is to perform Monte Carlo analyses, it is acceptable to perform preliminary runs to determine a suitable value for $|A_{Sample}|$ and to study the correlation of *ssd* and *essd*. If the correlation is high, *ssd* can be replaced with *essd*. A modification of the optimization algorithm is not necessary.

As a second objective, interference of the non-ideal part with the environment has to be avoided. Therefore, the optimization goal is to minimize the weighted sum of $g(\vec{x})$ and $ssd(A, B_A)$:

$$f(\vec{x}) = w_1 \cdot g(\vec{x}) + w_2 \cdot ssd(A, B_A)$$

The sets A and B_A have to be constructed from $\vec{x}$. If the summed square distance is estimated, the objective function is replaced by

$$f_{essd}(\vec{x}) = w_1 \cdot g(\vec{x}) + w_2 \cdot essd(A, B_A) \quad .$$

In both cases, the weights were chosen to $w_1 = 100,000$ and $w_2 = 1$. The weight w_1 was set to a comparatively high value due to the fact that colliding parts do not comply with real world restrictions. The optimiziation terminated after $10,000$ objective function evaluations.

The nodes of A_{Sample} should be chosen such that they are evenly distributed on the triangle mesh. This can, for instance, be obtained by drawing them uniformly at random from A, or by making use of the underlying (possibly sorted) data structure of the triangle mesh. In this case study, the latter method was used. It makes sense to choose A_{Sample} only once, at the beginning of the optimization, as otherwise, the optimization problem would be noisy, and strategies to cope with noisy environments had to be incorporated into the optimization algorithm. For more information on optimization in noisy environments, the reader is referred to the surveys of Beyer [Bey00] and Jin [Jin05].

For the correlation analysis, the *Pearson product-moment correlation coefficient*, which is a widely-used statistical measure, was computed in conjunction to a visual inspection. Pearson's correlation coefficient is defined as follows [RN88, OM88]:

$$r = \frac{\sum_{i=1}^{N}[(X_i - \overline{X}) \cdot (Y_i - \overline{Y})]}{\sqrt{\sum_{i=1}^{N}(X_i - \overline{X})^2 \cdot \sum_{i=1}^{N}(Y_i - \overline{Y})^2}} \tag{6.6}$$

where $\{X_1, X_1, \ldots, X_N\}$ and $\{Y_1, Y_2, \ldots, Y_N\}$ are observations, and $\overline{X}$ and $\overline{Y}$ are the respective sample means. The correlation coefficient r is bounded by $[-1, 1]$. Values near 1 indicate a strong positive correlation whereas values near -1 indicate a strong negative correlation.

Figure 6.7 summarizes the results of the ten optimization runs. The correlation of *ssd* and *essd* was investigated both visually and statistically. The two-dimensional plots already indicate that *ssd* and *essd* are strongly correlated. For each search space position $\vec{x}$ visited during an optimization run, the pair $(f(\vec{x}), f_{essd}(\vec{x}))$ is drawn. The values are very close to the curve $f(\vec{x}) = f_{essd}(\vec{x})$, which represents perfect linear correlation. Also, the correlation coefficient r is very close to 1 for each single optimization run. Both the statistical and the visual analysis show that *ssd* and *essd* are highly correlated in the given scenario. In order to speed up the optimization, *ssd* was replaced by *essd* in the following experiments.

Performance Comparison

The performance of different PSO algorithms was compared by using two test scenarios, both based on the assembly shown in Figure 6.2, and described above.

- *Zero test:* In order to get a first impression if PSO is suited for the relative positioning problem a simple test scenario with known global optimum was constructed. The non-ideal geometry is a rotated and translated version of the ideal crossbeam. As the shape is not modified, the non-ideal part also has 14,440 triangles. The objective function is $f(\vec{x}) = d'_H(A, B)$, where A (computed from $\vec{x}$) and B are the sets of triangle nodes of the non-ideal and ideal part, respectively. Due to the construction procedure of this test, the objective value of the global optimum is $f(\vec{x}^*) = 0$. Therefore, the test is called *Zero test*.

- *Crossbeam test:* A more realistic scenario was investigated in the so-called *Crossbeam test*. A non-ideal crossbeam with 49,030 triangles is to be positioned in the environment. The involved parts are described above and depicted in Figures 6.2 and 6.3. The objective function is given by

$$f(\vec{x}) = f_{essd}(\vec{x}) = w_1 \cdot g(\vec{x}) + w_2 \cdot essd(A, B_A) \ .$$

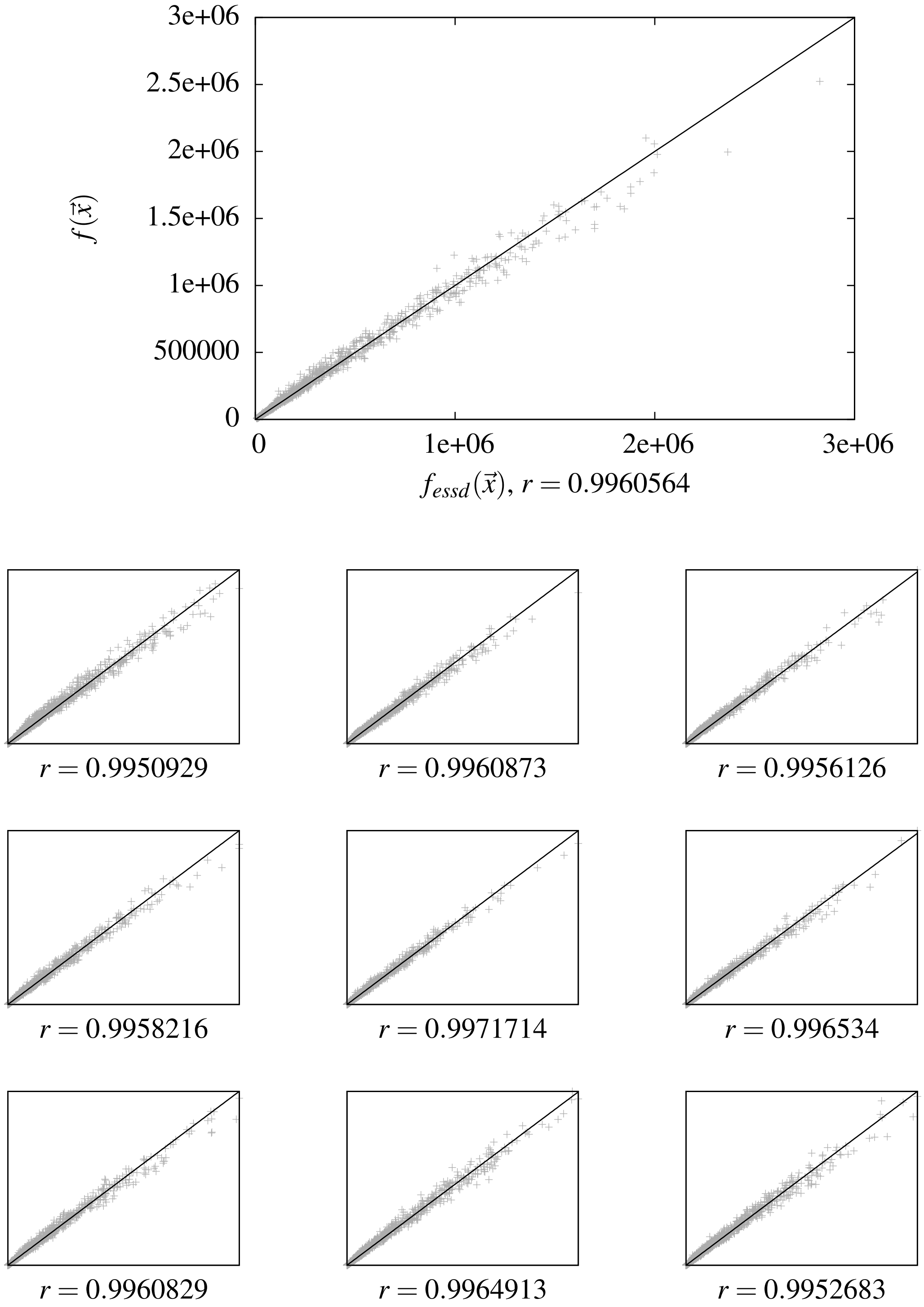


Figure 6.7: Visual and statistical inspection of the correlation of *ssd* and *essd* on the given assembly.

Table 6.1: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of different PSO variants on the relative positioning problem.

	Zero	Crossbeam
Hyperbolic	**2.7387e-06±3.3011e-07 (1.1615e-06)**	**8047.3±5.0875 (17.901)**
Infinity	9.9619e-06±1.6197e-06 (5.6994e-06)	8076±16.653 (58.596)
Reflect-Z	1.2377e-05±1.7385e-06 (6.1174e-06)	8089.9±14.018 (49.324)
Reflect-A	*2.1664e-04±3.2942e-05 (1.1591e-04)*	*8107.1±17.786 (62.583)*
MPSO-6	3.2131e-06±5.087e-07 (1.79e-06)	8058.3±8.905 (31.334)
MPSO-30	3.2566e-06±4.6569e-07 (1.6386e-06)	8059.8±11.213 (39.455)

where, again, the weights were chosen to $w_1 = 100,000$ and $w_2 = 1$, and $|A_{Sample}|$ was set to $|A_{Sample}| = 0.0025 \cdot |A|$. The same assembly and objective function was used in the above correlation analysis.

In both cases, the particle swarm optimizer was allowed to perform 10,000 function evaluations. Each experimental configuration was repeated 50 times. Table 6.1 shows mean values, respective 95% confidence intervals, and standard deviations of the final objective values found by different PSO configurations. Table 6.2 depicts the results of the Wilcoxon rank sum test. The following algorithms were tested: Hyperbolic and Reflect-Z, as these strategies performed best in the experiments presented in Chapter 4. Infinity, as it is a widely-used method, which was also proposed for standard PSO [BK07]. Additionally, the adaptive PSO algorithms introduced in Chapter 5 were included, to allow a performance comparison of the adaptive algorithms to standard particle swarm optimization:

- Reflect-A: PSO with velocity adaptation ($\rho = 0.2$) and Reflect-Z bound handling.
- MPSO-6: MPSO-1 with an adaptation interval of $k_1 = n = 6$ and Reflect-Z and Hyperbolic as subswarm strategies.
- MPSO-30: MPSO-1 with a larger adaptation interval of $k_1 = 30$ and Reflect-Z and Hyperbolic as subswarm strategies.

MPSO-1 and MPSO-2 produced solutions of similar quality in the experiments presented in Section 5.2.2 when using Reflect-Z and Hyperbolic as subswarm configurations. Due to its simpler design with less parameters to tune, MPSO-1 was chosen for the subsequent experiments. Different settings of the adaptation interval were tested.

All tested PSO algorithms reliably passed the *Zero test* by providing solutions very close to the global optimum (see Table 6.1). Slight performance differences are

Table 6.2: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows on which benchmarks A performed significantly better than B.

	1	2	3	4	5	6
Hyperbolic (1)	{}	{Zero, Crossb.}	{Zero, Crossb.}	{Zero, Crossb.}	{}	{}
Infinity (2)	{}	{}	{Crossb.}	{Zero, Crossb.}	{}	{}
Reflect-Z (3)	{}	{}	{}	{Zero}	{}	{}
Reflect-A (4)	{}	{}	{}	{}	{}	{}
MPSO-6 (5)	{}	{Zero, Crossb.}	{Zero, Crossb.}	{Zero, Crossb.}	{}	{}
MPSO-30 (6)	{}	{Zero, Crossb.}	{Zero, Crossb.}	{Zero, Crossb.}	{}	{}

already visible: Hyperbolic, MPSO-6, and MPSO-30 significantly outperformed the other variants (see Table 6.2).

In the *Crossbeam test*, the best performing algorithms were again Hyperbolic, MPSO-6, and MPSO-30, with no significant performance differences among each other (Table 6.2). The average objective values obtained by MPSO-6 and MPSO-30 are very similar (see Table 6.1), which means that the size of the adaptation interval hardly influenced the algorithm's performance. The bound handling strategies Infinity and Reflect-Z showed similar performance in both tests, with slight advantage for Infinity.

PSO with velocity adaptation performed significantly worse than most of the other algorithms. Most probably, the reason is the low number of function evaluations used in this application. The convergence plot given in Figure 6.8 shows that Reflect-A is considerably slower than the other algorithms, but still visibly improves the solution quality until the end of the optimization.

Summarized, all PSO algorithms were able to pass the *Zero test*, which indicates that particle swarm optimization is a good candidate to solve the relative positioning problem. Due to the low number of used function evaluations, PSO with velocity adaptation was not competitive to most of the other algorithms. The best performing algorithms in both the *Zero* and the *Crossbeam test* were Hyperbolic, MPSO-6 and MPSO-30.

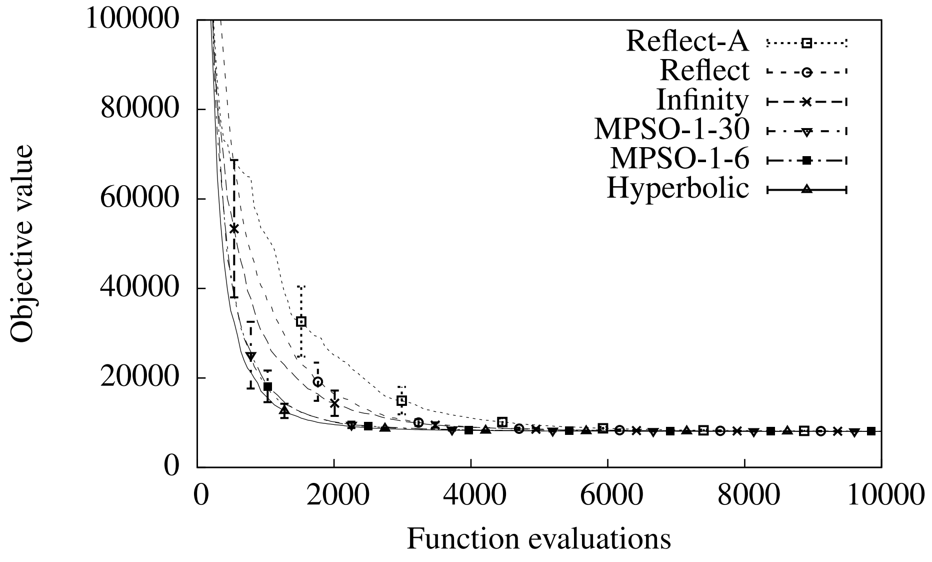


Figure 6.8: Convergence plot of the *Crossbeam test*. In the plot, average objective values are shown. Vertical bars depict the standard deviation.

7. Conclusion

In this thesis, fundamental theoretical and experimental results on bound-constrained particle swarm optimization were presented. It was shown that bound handling plays a crucial role when solving high-dimensional problems with PSO algorithms. Several strategies to cope with this fact were proposed, investigated in detail, and applied to practical problems.

In Chapter 3, it was proved mathematically that, if PSO is applied to box-constrained problems, many particles leave the feasible space at the beginning of the optimization, with overwhelming probability, even if particle velocities are initialized to zero. This theoretical result implies that the method how box constraints are dealt with (the so-called *bound handling strategy*) has significant impact on initial particle swarm behavior. Furthermore, by using a simplified PSO model, it was shown in Section 3.4 that the probability that a particle becomes infeasible is constant if the interval from which particle velocities are chosen is scaled with respect to the problem dimensionality.

The experimental evaluation presented in Chapter 4 confirmed the theoretical results, and additionally showed that bound handling not only has strong impact on initial particle swarm behavior, but also significantly influences the final solution quality achieved by particle swarm optimization, especially when solving high-dimensional problems.

Three ways to cope with these facts were proposed and investigated:

- *Careful design and selection of bound handling strategies.* In order to support the design and selection of bound handling strategies, a thorough experimental study of the strengths and weaknesses of thirteen commonly-used strategies was presented in Section 4.4.2.

- *Use of self-adaptation.* In Section 5.2 it was shown that bound handling can often be automatically adapted to the current optimization problem by using Multi-Swarm PSO with Migration.

- *Use of velocity adaptation.* Based on the theoretical results presented in Section 3.4, the concept of velocity adaptation was introduced in Section 5.3. The experimental evaluation showed that the use of velocity adaptation can reduce the impact of bound handling on particle swarm performance. At the same time, the solution quality was often improved significantly.

The most promising strategies from Section 4.4.2 as well as both adaptive algorithms were applied to the relative positioning problem in Chapter 6. The experimental results confirmed that bound handling has significant impact on the solution quality obtained by particle swarm optimization.

Directions of future research

The establishment of a broad theoretical basis for PSO algorithms is required in order to support practical PSO application. Much needed are results on expected solution quality and runtime of particle swarm optimization. Particle interaction is essential for the success of PSO algorithms, but has not yet been studied theoretically. A theoretical analysis of the impact of different neighborhood topologies on particle swarm performance could help users of PSO algorithms to select an appropriate topology for a given problem.

Adaptive and self-optimizing PSO algorithms are intended to reduce the necessity of manual parameter adjustment by an expert, and are therefore very useful for PSO applications. Currently, Multi-Swarm PSO with Migration is extended to not only use a finite number of predefined parameter sets but to allow more fine-grained parameter adaptation by using mutation techniques. First promising results are presented in [RHW10]. Further topics of future research are the design of new adaptation strategies, with focus on fine-grained parameter adaptation and simplicity, convergence and runtime analyses of these strategies, and the application of the new algorithms to multi-objective and combinatorial optimization problems.

A. Theoretical Derivations

A.1 Sum of Uniformly Distributed Random Variables

The probability density function of the sum of arbitrary many uniformly distributed random variables can be computed by using the following theorem of Bradley and Gupta [BG02, Theorem 1]:

Theorem A.1 (Bradley and Gupta (2002))**.** *The density of the sum of n independent random variables, uniformly distributed in the intervals $[c_j - a_j, c_j + a_j]$ for $j = 1, 2, \dots, n$ is given by*

$$f_n(x) = \frac{\sum\limits_{\vec{\varepsilon}\in\{-1,1\}^n} \left(x + \sum\limits_{j=1}^{n}(\varepsilon_j a_j - c_j)\right)^{n-1} \cdot \operatorname{sign}\left(x + \sum\limits_{j=1}^{n}(\varepsilon_j a_j - c_j)\right) \prod\limits_{j=1}^{n} \varepsilon_j}{(n-1)!2^{n+1}\prod\limits_{j=1}^{n} a_j} ,$$

in which the sum is over all 2^n vectors of signs

$$\vec{\varepsilon} = (\varepsilon_1, \varepsilon_2, \dots, \varepsilon_n) \in \{-1,1\}^n \quad \textit{i.e. each } \varepsilon_j = \pm 1$$

and

$$\operatorname{sign}(y) = \begin{cases} 1 & \text{if } y > 0 \\ 0 & \text{if } y = 0 \\ -1 & \text{if } y > 0 \ . \end{cases}$$

A.2 Particle Explosion – Uniform Velocity Initialization

In the following, the detailed derivation of Equation (3.6) of part (ii) of the proof of Theorem 3.3, is presented. In part (ii) of the proof of Theorem 3.3, the d-th component of the position vector $\vec{x}_{i,1}$ is rewritten as $x_{i,1,d} = k_3 + k_4 + k_5$ with $k_3 = \omega \cdot v_{i,0,d}$, $k_4 = (1 - c_2 r_{2,i,1,d}) \cdot x_{i,0,d}$, and $k_5 = c_2 \cdot r_{2,i,1,d} \cdot l_{i,0,d}$. As particles and velocities are initialized uniformly at random in $\mathcal{S} = [-r, r]^n$, k_3 and k_5 are distributed uniformly at

random in $[-\omega r, \omega r]$ and $[-c_2 r_{2,i,1,d} r, c_2 r_{2,i,1,d} r]$, respectively. We distinguish seven cases.
Case 1: $r_{2,i,1,d} = 0$
If $r_{2,i,1,d} = 0$, the d-th component of particle i's position vector is

$$x_{i,1,d} = \omega v_{i,0,d} + x_{i,0,d}$$

From part (i) of the proof we know that the probability $p_{case1}(\omega)$ that the d-th search space dimension is violated computes to $p_{case1}(\omega) = \frac{\omega}{4}$.
Case 2: $0 < r_{2,i,1,d} < \frac{\omega}{2c_2}$
With our assumptions, the following inequation is true: $r_{2,i,1,d} < \frac{\omega}{2c_2} < \frac{1}{c_2}$. Therefore, k_4 is distributed uniformly at random in $[-(1 - c_2 r_{2,i,1,d})r, (1 - c_2 r_{2,i,1,d})r]$. The density function of the sum of arbitrary many, non-identically and independently distributed uniform random variables can be computed using the formula presented by Bradley and Gupta [BG02, Theorem 1]. Let $f_{x_{i,1,d}}(z)$ be the density function of $x_{i,1,d}$, which can be computed by using a modern computer algebra system. Then, with our assumptions about ω and c_2, the probability $p_{case2}(r_{2,i,1,d}, c_2, \omega)$ that particle i violates the d-th search space boundary is given by

$$\begin{aligned} p_{case2}(r_{2,i,1,d}, c_2, \omega) &= \int_{-\infty}^{-r} f_{x_{i,1,d}}(z)\mathrm{d}z + \int_{r}^{\infty} f_{x_{i,1,d}}(z)\mathrm{d}z \\ &= \frac{-3\omega^2 + 6c_2 r_{2,i,1,d}\omega - 4c_2^2 r_{2,i,1,d}^2}{-12\omega(1 - c_2 r_{2,i,1,d})} \quad . \end{aligned}$$

Case 3: $\frac{\omega}{2c_2} \le r_{2,i,1,d} < \frac{2-\omega}{2c_2}$
We yield $r_{2,i,1,d} < \frac{2-\omega}{2c_2} < \frac{1}{c_2}$, which implies that the term k_4 is distributed uniformly at random in $[-(1 - c_2 r_{2,i,1,d})r, (1 - c_2 r_{2,i,1,d})r]$. With the formula of Bradley and Gupta, the probability $p_{case3}(r_{2,i,1,d}, c_2, \omega)$ that a single boundary is exceeded evaluates to

$$\begin{aligned} p_{case3}(r_{2,i,1,d}, c_2, \omega) &= \int_{-\infty}^{-r} f_{x_{i,1,d}}(z)\mathrm{d}z + \int_{r}^{\infty} f_{x_{i,1,d}}(z)\mathrm{d}z \\ &= \frac{\omega^2}{24(1 - c_2 r_{2,i,1,d})c_2 r_{2,i,1,d}} \quad . \end{aligned}$$

Case 4: $\frac{2-\omega}{2c_2} \le r_{2,i,1,d} < \frac{1}{c_2}$
Again, k_4 is distributed uniformly at random in $[-(1 - c_2 r_{2,i,1,d})r, (1 - c_2 r_{2,i,1,d})r]$. The probability $p_{case4}(r_{2,i,1,d}, c_2, \omega)$ that particle i crosses the boundary in dimension d computes to

$$p_{case4}(r_{2,i,1,d}, c_2, \omega) = \frac{4c_2^2 r_{2,i,1,d}^2 + 6\omega c_2 r_{2,i,1,d} - 8c_2 r_{2,i,1,d} + 3\omega^2 + 4 - 6\omega}{12\omega c_2 r_{2,i,1,d}}$$

Case 5: $r_{2,i,1,d} = \frac{1}{c_2}$
In this case, $x_{i,1,d}$ is given by $x_{i,1,d} = \omega v_{i,0,d} + l_{i,0,d}$. According to part (i) of the proof, the probability that particle i leaves the search space in dimension d is $\frac{\omega}{4}$. Remember that Assumption 3.3 states that $l_{i,0}$ is distributed uniformly at random in $\mathcal{S}$.
Case 6: $\frac{1}{c_2} < r_{2,i,1,d} < \frac{2+\omega}{2c_2}$
As $r_{2,i,1,d} > \frac{1}{c_2}$, k_4 is now distributed uniformly at random in $[(1-c_2 r_{2,i,1,d})r, -(1-c_2 r_{2,i,1,d})r]$, and the probability $p_{case6}(r_{2,i,1,d}, c_2, \omega)$ that particle i exceeds a boundary is

$$p_6(r_{2,i,1,d}, c_2, \omega) = \frac{4c_2^2 r_{2,i,1,d}^2 + 6\omega c_2 r_{2,i,1,d} - 8c_2 r_{2,i,1,d} + 3\omega^2 + 4 - 6\omega}{12\omega c_2 r_{2,i,1,d}}$$

Case 7: $\frac{2+\omega}{2c_2} \le r_{2,i,1,d} \le 1$
In this case, k_4 is distributed uniformly at random in $[(1-c_2 r_{2,i,1,d})r, -(1-c_2 r_{2,i,1,d})r]$, and the probablity $p_{case7}(r_{2,i,1,d}, c_2, \omega)$ that a particle violates the d-th search space bound evaluates to

$$p_{case7}(r_{2,i,1,d}, c_2, \omega) = \frac{24 + \omega^2 + 24c_2^2 r_{2,i,1,d}^2 - 48c_2 r_{2,i,1,d}}{-24c_2 r_{2,i,1,d}(1 - c_2 r_{2,i,1,d})} \quad .$$

Summarized, the probability $q_1(r_{2,i,1,d}, c_2, \omega)$ that particle i crosses the search space bound in dimension d, i.e., the probability that $x_{i,1,d} \notin [-r, r]$, is computed to:

$$q_1(r_{2,i,1,d}, c_2, \omega) = \int_{-\infty}^{-r} f_{x_{i,1,d}}(z)\mathrm{d}z + \int_{r}^{\infty} f_{x_{i,1,d}}(z)\mathrm{d}z$$
$$= \begin{cases} \frac{-3\omega^2 + 6c_2 r_{2,i,1,d}\omega - 4c_2^2 r_{2,i,1,d}^2}{-12\omega(1-c_2 r_{2,i,1,d})} & \text{if } 0 \le r_{2,i,1,d} < \frac{\omega}{2c_2} \\ \frac{\omega^2}{24(1-c_2 r_{2,i,1,d})c_2 r_{2,i,1,d}} & \text{if } \frac{\omega}{2c_2} \le r_{2,i,1,d} < \frac{2-\omega}{2c_2} \\ \frac{4c_2^2 r_{2,i,1,d}^2 + 6\omega c_2 r_{2,i,1,d} - 8c_2 r_{2,i,1,d} + 3\omega^2 + 4 - 6\omega}{12\omega c_2 r_{2,i,1,d}} & \text{if } \frac{2-\omega}{2c_2} \le r_{2,i,1,d} < \frac{2+\omega}{2c_2} \\ \frac{24 + \omega^2 + 24c_2^2 r_{2,d}^2 - 48c_2 r_{2,i,1,d}}{-24c_2 r_{2,i,1,d}(1-c_2 r_{2,i,1,d})} & \text{if } \frac{2+\omega}{2c_2} \le r_{2,i,1,d} \le 1 \end{cases}$$

A.3 Particle Explosion – Half-diff Velocity Initialization

In the following, the detailed derivation of Equation (3.11) of part (ii) of the proof of Theorem 3.9, is presented. Let particle i be an arbitrary particle that satisfies the given assumptions. Its position and velocity in the first iteration compute to

$$\begin{aligned} v_{i,1,d} =& \omega \cdot v_{i,0,d} + c_2 \cdot r_{2,i,1,d} \cdot (l_{i,0,d} - x_{i,0,d}) \\ x_{i,1,d} =& x_{i,0,d} + v_{i,1,d} = \underbrace{\tfrac{\omega}{2} \cdot z_{i,d}}_{k_8} + \underbrace{c_2 \cdot r_{2,i,1,d} \cdot l_{i,0,d}}_{k_9} + \underbrace{\left(1 - \tfrac{\omega}{2} - c_2 \cdot r_{2,i,1,d}\right) \cdot x_{i,0,d}}_{k_{10}} \end{aligned}$$

for $d = 1, \dots, n$.
Case 1: $0 \leq r_{2,i,1,d} \leq \frac{2-\omega}{2c_2}$
Since

$$r_{2,i,1,d} \leq \frac{2-\omega}{2c_2} \quad \Leftrightarrow \quad 1 - \frac{\omega}{2} - c_2 \cdot r_{2,i,1,d} \geq 0$$

we compute

$$(1)\ x_{i,1,d} \leq \tfrac{\omega}{2} \cdot r + c_2 \cdot r_{2,i,1,d} \cdot r + (1 - \tfrac{\omega}{2} - c_2 \cdot r_{2,i,1,d}) \cdot r = r$$

$$(2)\ x_{i,1,d} \geq \tfrac{\omega}{2} \cdot (-r) + c_2 \cdot r_{2,i,1,d} \cdot (-r) + (1 - \tfrac{\omega}{2} - c_2 \cdot r_{2,i,1,d}) \cdot (-r) = -r \quad .$$

This means that particle i does not leave the search space in dimension d if $0 \leq r_{2,i,1,d} \leq \frac{2-\omega}{2c_2}$ holds.
Case 2: $\frac{2-\omega}{2c_2} < r_{2,i,1,d} < \frac{1}{c_2}$ or $\frac{1}{c_2} < r_{2,i,1,d} \leq 1$
Similar to the proof of Theorem 3.3, part (ii), $x_{i,1,d}$ is rewritten as the sum of three stochastic variables: $x_{i,1,d} = k_8 + k_9 + k_{10}$. The initial particle positions as well as the random vectors z_i, $i = 1, \dots, m$, are distributed uniformly at random in $\mathcal{S}$. Assumption 3.3 states that the same holds for the local guides $\vec{l}_{i,0}$, $i = 1, \dots, m$. Hence, k_8 and k_9 are distributed uniformly at random in $[-\frac{\omega}{2}r, \frac{\omega}{2}r]$ and $[-c_2 r_{2,i,1,d} r, c_2 r_{2,i,1,d} r]$, respectively. From $r_{2,i,1,d} > (2-\omega)/(2c_2)$, $1 - \omega/2 - c_2 r_{2,i,1,d} < 0$ follows, and therefore k_{10} is distributed uniformly at random in $[(1 - \frac{\omega}{2} - c_2 r_{2,i,1,d})r, (-1 + \frac{\omega}{2} + c_2 r_{2,i,1,d})r]$. This means that $x_{i,1,d}$ is the sum of three stochastic variables that are distributed uniformly at random in their respective intervals. The probability density function $f_{x_{i,1,d}}$ of $x_{i,1,d}$ can be computed by using the appraoch of Bradley and Gupta [BG02, Theorem 1]. The probability $q_3(r_{2,i,1,d}, c_2, \omega)$ that particle i violates the d-th search space bound is then computed to

$$q_3(r_{2,i,1,d}, c_2, \omega) = \int_{-\infty}^{-r} f_{x_{i,1,d}}(z)\mathrm{d}z + \int_{r}^{\infty} f_{x_{i,1,d}}(z)\mathrm{d}z$$

$$= \begin{cases} (6\omega c_2 r_{2,i,1,d}(2-\omega-2c_2 r_{2,i,1,d}))^{-1}(-\omega^3 - 24 c_2 r_{2,i,1,d} & \\ \quad +24 c_2^2 r_{2,i,1,d}^2 + 24\omega c_2 r_{2,i,1,d} + 8 - 8 c_2^3 r_{2,i,1,d}^3 - 12\omega & \\ \quad +6\omega^2 - 6\omega^2 c_2 r_{2,i,1,d} - 12\omega c_2^2 r_{2,i,1,d}^2) & \text{if } \frac{2-\omega}{2c_2} < r_{2,i,1,d} < \frac{1}{c_2} \\ \dfrac{-\omega^3 + 24\omega c_2 r_{2,i,1,d} - 12\omega + 6\omega^2 - 6\omega^2 c_2 r_{2,i,1,d} - 12\omega c_2^2 r_{2,i,1,d}^2}{6\omega c_2 r_{2,i,1,d}\left(2-\omega-2c_2 r_{2,i,1,d}\right)} & \text{if } \frac{1}{c_2} < r_{2,i,1,d} \leq 1 \end{cases}$$

Case 3: $r_{2,i,1,d} = \frac{1}{c_2}$
Particle i's position in the first iteration is given by $x_{i,1,d} = \frac{\omega}{2} z_{i,d} + l_{i,0,d} - \frac{\omega}{2} x_{i,0,d}$. Hence, $x_{i,1,d}$ is the sum of three random variables that are distributed uniformly at random in $[-\frac{\omega}{2} r, \frac{\omega}{2} r]$ and $[-r,r]$, respectively. The density function $f_{x_{i,1,d}}$ can again be computed according to Bradley and Gupta [BG02, Theorem 1]. The probability $q_3(r_{2,i,1,d}, c_2, \omega)$ that a particle exceeds the d-th search space bound is

$$q_3(r_{2,i,1,d}, c_2, \omega) = \int_{-r-\omega r}^{-r} f_{x_{i,1,d}}(z)\mathrm{d}z + \int_{r}^{r+\omega r} f_{x_{i,1,d}}(z)\mathrm{d}z = \frac{\omega}{6} \ .$$

Summarizing the three cases, Equation (3.11) is obtained:

$$q_3(r_{2,i,1,d}, c_2, \omega) = \int_{-\infty}^{-r} f_{x_{i,1,d}}(z)\mathrm{d}z + \int_{r}^{\infty} f_{x_{i,1,d}}(z)\mathrm{d}z$$
$$= \begin{cases} 0 & \text{if } 0 \le r_{2,i,1,d} \le \frac{2-\omega}{2c_2} \\ (6\omega c_2 r_{2,i,1,d}(2-\omega-2c_2 r_{2,i,1,d}))^{-1}(-\omega^3 - 24c_2 r_{2,i,1,d} \\ \quad +24c_2^2 r_{2,i,1,d}^2 + 24\omega c_2 r_{2,i,1,d} + 8 - 8c_2^3 r_{2,i,1,d}^3 - 12\omega \\ \quad +6\omega^2 - 6\omega^2 c_2 r_{2,i,1,d} - 12\omega c_2^2 r_{2,i,1,d}^2) & \text{if } \frac{2-\omega}{2c_2} < r_{2,i,1,d} < \frac{1}{c_2} \\ \frac{\omega}{6} & \text{if } r_{2,i,1,d} = \frac{1}{c_2} \\ \frac{-\omega^3 + 24\omega c_2 r_{2,i,1,d} - 12\omega + 6\omega^2 - 6\omega^2 c_2 r_{2,i,1,d} - 12\omega c_2^2 r_{2,i,1,d}^2}{6\omega c_2 r_{2,i,1,d}(2-\omega-2c_2 r_{2,i,1,d})} & \text{if } \frac{1}{c_2} < r_{2,i,1,d} \le 1 \end{cases}$$

A.4 Particle Explosion – Half-diff Velocity Initialization, Part 2

The function $q_3(r_{2,i,1,d}, c_2, \omega)$ (see above or refer to Equation (3.11)) is continuous for

(1) $\frac{2-\omega}{2c_2} < r_{2,i,1,d} < \frac{1}{c_2}$,

(2) $\frac{1}{c_2} < r_{2,i,1,d} \le 1$

as sum $f+g$ and product $f \cdot g$ of continuous functions $f, g : I \to \mathbb{R}$ are again continuous in I, and the function $h = f/g : I' \to \mathbb{R}$ with $I' = \{x \in I : g(x) \neq 0\}$ is continuous in I' if f and g are continuous [For01, page 95]. From $r_{2,i,1,d} > \frac{2-\omega}{2c_2}$, $2 - w - 2c_2 r_{2,i,1,d} < 0$ follows. Additionally, $r_{2,i,1,d} > \frac{2-\omega}{2c_2} > 0$, $w > 0$, and $c_2 > 0$.

Hence, $6\omega c_2 r_{2,i,1,d}(2 - w - 2c_2 r_{2,i,1,d}) < 0$, and $q_3(r_{2,i,1,d}, c_2, \omega)$ is continuous in the given intervals.

Furthermore, it has to be shown that $q_3(r_{2,i,1,d}, c_2, \omega)$ is continuous at $r_{2,i,1,d} = 1/c_2$. From Equation (3.11), $q_3(1/c_2, c_2, \omega) = \omega/6$ is obtained. With an appropriate computer algebra system, the following left- and right-hand limits are computed, by using Equation (3.11):

$$\lim_{r_{2,i,1,d} \nearrow \frac{1}{c_2}} q_3(r_{2,i,1,d}, c_2, \omega) = \frac{\omega}{6}$$
$$\lim_{r_{2,i,1,d} \searrow \frac{1}{c_2}} q_3(r_{2,i,1,d}, c_2, \omega) = \frac{\omega}{6}$$

B. Experimental Results

B.1 The Wilcoxon Rank Sum Test: Examples

In this section, the concept of the one-sided Wilcoxon rank sum test is demonstrated by computing two small examples by hand. Two sample sets A and B, each of size $N = 3$, and the hypotheses as stated in Equations (4.1) and (4.2) are considered (repeated here for convenience):

$$H_0 : P(X_A < X_B) \leq \frac{1}{2}$$
$$H_1 : P(X_A < X_B) > \frac{1}{2}$$

The significance level is set to $\alpha = 0.08$.

Example 1

Let $A = \{0.5, 2, 2.9\}$ and $B = \{1.1, 2.8, 20\}$ be the two sample sets. The rank sets of samples A and B are $\{1,3,5\}$ and $\{2,4,6\}$, respectively. All possible rank sets for $N = 3$ with corresponding rank sums are given in Table B.1 (left). The rank sum of set A is $W = 9$, and the p-value is given by

$$p = \sum_{i=(N+1)\cdot N/2}^{W} p_i = \sum_{i=6}^{9} p_i = \frac{1}{20} + \frac{1}{20} + \frac{2}{20} + \frac{3}{20} = \frac{7}{20} = 0.35 \quad .$$

The values for p_i were derived from Table B.1 (left), and are shown in Table B.1 (right). Remember that each rank set has the same probability if the null hypothesis was true. Since

$$p = 0.35 \geq \alpha = 0.08$$

the null hypothesis is not rejected.

Example 2

As a second example, let $A = \{1, 30, 31\}$ and $B = \{100, 105, 110\}$ be the two sample sets. The rank sets of samples A and B are $\{1,2,3\}$ and $\{4,5,6\}$, respectively. Hence, $W = 6$, and the p-value is computed to $p = 1/20 = 0.05$. The null hypothesis is rejected due to the fact that $p = 0.05 < \alpha = 0.08$ holds.

Table B.1: Left: Possible rank sets and corresponding rank sums for one sample set, assuming two sample sets without ties of size $N = 3$ each. Right: Frequency and probablity of the occuring rank sums.

{1,2,3} **6**	{1,2,4} **7**	{1,2,5} **8**	{1,2,6} **9**
	{1,3,4} **8**	{1,3,5} **9**	{1,3,6} **10**
	{1,4,5} **10**	{1,4,6} **11**	{1,5,6} **12**
	{2,3,4} **9**	{2,3,5} **10**	{2,3,6} **11**
	{2,4,5} **11**	{2,4,6} **12**	{2,5,6} **13**
{3,4,5} **12**	{3,4,6} **13**	{3,5,6} **14**	{4,5,6} **15**

Rank sum	**Freq.**	**Prob.**
6	1	$p_6 = 1/20$
7	1	$p_7 = 1/20$
8	2	$p_8 = 2/20$
9	3	$p_9 = 3/20$
10	3	$p_{10} = 3/20$
11	3	$p_{11} = 3/20$
12	3	$p_{12} = 3/20$
13	2	$p_{13} = 2/20$
14	1	$p_{14} = 1/20$
15	1	$p_{15} = 1/20$

B.2 Velocity Initialization

Detailed results for the experimental comparison of different velocity initialization strategies (Section 4.3):

- *Nearest-Z* bound handling
  - Sample means, 95% confidence intervals, sample standard deviations of $N = 100$ runs: Table B.2
- *Random-Z* bound handling
  - Sample means, 95% confidence intervals, sample standard deviations of $N = 100$ runs: Table B.3

Table B.2: Sample mean of final objective values, 95% confidence intervals, and sample standard deviations (in brackets) of different velocity initialization strategies on various 100-dimensional benchmarks. The best objective values are presented together with the function name. *Nearest-Z* bound handling was utilized.

	Sphere (0)	Rosenbrock (0)
Uniform	**5.9444e-06±1.6646e-07 (8.3891e-07)**	*265.98±84.995 (428.36)*
Zero	*6.0396e-06±1.9111e-07 (9.6314e-07)*	227.98±59.925 (302.01)
Half-diff	5.9515e-06±1.6588e-07 (8.3601e-07)	**200.52±15.094 (76.069)**
	Ackley (0)	Griewank (0)
Uniform	*2.1891±0.40483 (2.0402)*	*5.9213e-03±2.4547e-03 (0.012371)*
Zero	1.8254±0.1251 (0.63049)	4.6964e-03±2.1546e-03 (0.010859)
Half-diff	**1.667±0.12876 (0.64891)**	**4.5718e-03±2.1514e-03 (0.010842)**
	Rastrigin (0)	Schwefel (≈ -41898.3)
Uniform	*441.37±13.379 (67.426)*	*-28123±311.46 (1569.7)*
Zero	428.15±10.722 (54.036)	-28277±313.79 (1581.4)
Half-diff	**366.22±7.7675 (39.146)**	**-28437±309.36 (1559.1)**
	f1 (-450)	f2 (-450)
Uniform	*-443.36±11.326 (57.082)*	*48740±3732.1 (18809)*
Zero	**-450 (0)**	42771±2793.5 (14079)
Half-diff	-447.36±5.2478 (26.447)	**41490±2436.5 (12279)**
	f3 (-450)	f5 (-310)
Uniform	*58204000±3975800 (20037000)*	**28904±882.66 (4448.4)**
Zero	57085000±3306500 (16664000)	*29679±886.91 (4469.8)*
Half-diff	**52624000±3151500 (15883000)**	29180±765.47 (3857.8)
	f6 (390)	f8 (-140)
Uniform	702240±1392200 (7016500)	**-118.71±6.4417e-03 (0.032465)**
Zero	**402210±771430 (3887800)**	*-118.7±5.4789e-03 (0.027612)*
Half-diff	*1347200±2624700 (13228000)*	**-118.71±5.9884e-03 (0.03018)**
	f9 (-330)	f10 (-330)
Uniform	**24.989±11.058 (55.73)**	**42.266±12.782 (64.417)**
Zero	31.861±11.847 (59.707)	42.604±11.496 (57.938)
Half-diff	*32.932±10.413 (52.481)*	*49.191±12.657 (63.786)*
	f11 (-460)	f12 (90)
Uniform	*223.97±1.1159 (5.6241)*	*1367400±106380 (536150)*
Zero	**222.42±1.0272 (5.1767)**	**1097200±107580 (542180)**
Half-diff	222.97±1.081 (5.4479)	1135300±124970 (629840)
	f13 (-130)	f14 (-300)
Uniform	-65.341±3.0503 (15.373)	*-253.25±0.081793 (0.41222)*
Zero	*-62.272±3.1644 (15.948)*	-253.37±0.084162 (0.42416)
Half-diff	**-65.684±2.6176 (13.192)**	**-253.41±0.090291 (0.45504)**

Table B.3: Sample mean of final objective values, 95% confidence intervals, and sample standard deviations (in brackets) of different velocity initialization strategies on various 100-dimensional benchmarks. The best objective values are presented together with the function name. *Random-Z* bound handling was utilized.

	Sphere (0)	Rosenbrock (0)
Uniform	*6.1541e-06±1.6062e-07 (8.0949e-07)*	**197.24±10.137 (51.089)**
Zero	**6.1236e-06±1.9575e-07 (9.8652e-07)**	197.58±10.408 (52.456)
Half-diff	6.132e-06±1.6757e-07 (8.445e-07)	*201.9±9.2629 (46.683)*
	Ackley (0)	Griewank (0)
Uniform	*1.6438±0.15692 (0.79082)*	7.3293e-03±7.2298e-03 (0.036436)
Zero	1.5961±0.15027 (0.75732)	*7.4434e-03±3.5067e-03 (0.017673)*
Half-diff	**1.5611±0.14517 (0.73164)**	**3.7379e-03±1.9864e-03 (0.010011)**
	Rastrigin (0)	Schwefel (≈ -41898.3)
Uniform	*261.3±5.904 (29.755)*	**-25772±230.25 (1160.4)**
Zero	260.44±6.5915 (33.219)	-25588±261.9 (1319.9)
Half-diff	**259.75±6.483 (32.673)**	*-25510±232.94 (1174)*
	f1 (-450)	f2 (-450)
Uniform	**-450 (0)**	24432±1322.9 (6667.1)
Zero	**-450 (0)**	**23394±1164.5 (5868.9)**
Half-diff	**-450 (0)**	*25170±1732.2 (8729.9)*
	f3 (-450)	f5 (-310)
Uniform	**72747000±6512200 (3.282e+07)**	31642±716.64 (3611.7)
Zero	*80919000±7246900 (36522000)*	**31403±654.21 (3297.1)**
Half-diff	7.557e+07±5833300 (29399000)	*31841±675.46 (3404.2)*
	f6 (390)	f8 (-140)
Uniform	*593.64±10.85 (54.682)*	**-118.71±6.4568e-03 (0.032541)**
Zero	**576.43±10.349 (52.158)**	*-118.7±6.404e-03 (0.032275)*
Half-diff	587.45±10.702 (53.936)	*-118.7±5.5891e-03 (0.028168)*
	f9 (-330)	f10 (-330)
Uniform	**44.919±10.872 (54.793)**	**87.858±14.042 (70.767)**
Zero	52.71±11.46 (57.755)	99.153±13.98 (70.457)
Half-diff	*59.29±10.966 (55.267)*	*102.2±14.465 (72.899)*
	f11 (-460)	f12 (90)
Uniform	*223.74±1.0878 (5.4821)*	374600±33194 (167290)
Zero	**222.75±1.0805 (5.4457)**	*377540±38288 (192960)*
Half-diff	222.81±1.2242 (6.1699)	**356330±32520 (163900)**
	f13 (-130)	f14 (-300)
Uniform	**-67.068±3.1907 (16.08)**	*-253.42±0.086922 (0.43807)*
Zero	-65.287±2.6727 (13.47)	**-253.45±0.099929 (0.50362)**
Half-diff	*-64.966±2.9668 (14.952)*	-253.43±0.090736 (0.45729)

B.3 Bound Handling

Detailed results for the experimental comparison of different bound handling strategies (Sections 4.4.1 and 4.4.2).

Results of Wilcoxon Rank Sum Test

Table B.4: Summary of one-sided Wilcoxon rank sum test with significance level $\alpha = 0.01$ for the two-dimensional benchmarks. For each algorithmic combination (A, B), this matrix shows on which benchmarks A performed significantly better than B.

2D	1	2	3	4	5	6	7
Hyperbolic (1)	{}	{}	{}	{f5}	{f11}	{}	{}
RandomBack (2)	{Schwefel, f8}	{}	{}	{f5, f8}	{}	{}	{f8}
Nearest-Z (3)	{Ackley, Rastrigin, Schwefel, f8}	{}	{}	{f5, f8}	{}	{}	{f8}
Random-Z (4)	{Schwefel, f8}	{}	{}	{}	{}	{}	{}
Reflect-Z (5)	{Schwefel, f8}	{}	{}	{f5, f8}	{}	{}	{f8}
Infinity (6)	{Schwefel, f8}	{}	{}	{f5, f8}	{f11}	{}	{}
Infinity-C (7)	{Schwefel, f8}	{}	{}	{f5, f8}	{}	{}	{}

Table B.5: Summary of one-sided Wilcoxon rank sum test with significance level $\alpha = 0.01$ for the two-dimensional benchmarks. For each algorithmic combination (A, B), this matrix shows how often A performed significantly better than B.

2D	1	2	3	4	5	6	7
Hyperbolic (1)	0	0	0	1	1	0	0
RandomBack (2)	2	0	0	2	0	0	1
Nearest-Z (3)	4	0	0	2	0	0	1
Random-Z (4)	2	0	0	0	0	0	0
Reflect-Z (5)	2	0	0	2	0	0	1
Infinity (6)	2	0	0	2	1	0	0
Infinity-C (7)	2	0	0	2	0	0	0

Table B.6: Summary of one-sided Wilcoxon rank sum test with significance level $\alpha = 0.01$ for the 30-dimensional benchmarks. For each algorithmic combination (A, B), this matrix shows how often A performed significantly better than B.

30D	1	2	3	4	5	6	7
Hyperbolic (1)	0	7	7	5	5	7	4
RandomBack (2)	4	0	4	2	1	5	3
Nearest-Z (3)	4	3	0	2	0	6	4
Random-Z (4)	5	3	5	0	2	9	4
Reflect-Z (5)	4	5	7	3	0	7	4
Infinity (6)	4	3	3	1	1	0	0
Infinity-C (7)	4	6	5	1	3	8	0

Table B.7: Summary of one-sided Wilcoxon rank sum test with significance level $\alpha = 0.01$ for the 30-dimensional benchmarks. For each algorithmic combination (A, B), this matrix shows on which benchmarks A performed significantly better than B. $\mathcal{B}$ denotes the set of all 18 benchmark functions.

30D	1	2	3	4	5	6	7
Hyperbolic (1)	{}	{Ra, f2, f3, f11, f12, f13, f14}	{Ra, f2, f3, f11, f12, f13, f14}	{Ra, f3, f5, f11, f13}	{Ra, f3, f11, f12, f13}	{Ra, f3, f5, f11, f12, f13, f14}	{Ra, f3, f11, f13}
Random Back (2)	{Schw, f6, f9, f10}	{}	{Ra, f2, f3, f12}	{Schw, f5}	{Ra}	{Ra, Schw, f5, f9, f10}	{Schw, f9, f10}
Nearest-Z (3)	{Schw, f6, f9, f10}	{Ackley, Schw, f11}	{}	{Schw, f5}	{}	{Schw, f5, f9, f10, f11, f13}	{Schw, f5, f9, f10}
Random-Z (4)	{Schw, f6, f9, f10, f12}	{f11, f12, f14}	{Ra, f2, f3, f12, f14}	{}	{Ra, f12}	{Ra, Schw, f5, f9, f10, f11, f12, f13, f14}	{Schw, f9, f10, f12}
Reflect-Z (5)	{Schw, f6, f9, f10}	{Schw, f10, f11, f13, f14}	{Schw, f2, f3, f9, f10, f12, f14}	{Schw, f5, f10}	{}	{Schw, f5, f9, f10, f11, f13, f14}	{Schw, f5, f9, f10}
Infinity (6)	{Schw, f6, f9, f10}	{f2, f3, f12}	{f2, f3, f12}	{f3}	{f3}	{}	{}
Infinity-C (7)	{Schw, f6, f9, f10}	{Ra, f2, f3, f11, f12, f14}	{Ra, f2, f3, f12, f14}	{f3}	{Ra, f3, f12}	{Ra, Schw, f5, f9, f10, f11, f13, f14}	{}

Table B.8: Summary of one-sided Wilcoxon rank sum test with significance level $\alpha = 0.01$ for the 100-dimensional benchmarks. For each algorithmic combination (A, B), this matrix shows on which benchmarks A performed significantly better than B. $\mathcal{B}$ denotes the set of all 18 benchmark functions.

100D	1	2	3	4	5	6	7
Hyperbolic (1)	{}	{Ro, Ack, Grie, Ra, f2, f3, f5, f6, f11, f12, f13, f14}	{Ro, Ack, Ra, f2, f3, f5, f6, f11, f12, f13, f14}	{Ro, Ack, Ra, f2, f3, f5, f11, f12, f13, f14}	{Ro, Ack, Ra, f2, f3, f11, f12, f13, f14}	$\mathcal{B}$	{Ro, Ack, Ra, Schw, f2, f3, f5, f8, f11, f12, f13, f14}
Random Back (2)	{Schw, f9, f10}	{}	{Ra, f2, f3, f12}	{Schw, f3, f5, f9}	{Ra}	$\mathcal{B}$	{Schw, f5, f9, f10}
Nearest-Z (3)	{Schw, f9, f10}	{Schw, f10, f11, f13, f14}	{}	{Schw, f3, f5, f9, f10}	{}	$\mathcal{B}$	{Schw, f5, f8, f9, f10}
Random-Z (4)	{Schw, f9, f10}	{Grie, Ra, f2, f11, f12, f13, f14}	{Ra, f2, f12}	{}	{Ra, f12}	$\mathcal{B}$	{Schw, f5, f9, f10}
Reflect-Z (5)	{Schw, f9, f10}	{Ro, Schw, f2, f5, f9, f10, f11, f13, f14}	{Schw, f2, f3, f5, f9, f10, f11, f12}	{Ro, Schw, f3, f5, f9, f10, f11}	{}	$\mathcal{B}$	{Schw, f5, f8, f9, f10, f11}
Infinity (6)	{}	{}	{}	{}	{}	{}	{}
Infinity-C (7)	{f9, f10}	{Ra, f2, f3, f11, f12, f13, f14}	{Ra, f2, f3, f12}	{f2, f3}	{Ra, f2, f3, f12}	$\mathcal{B}$	{}

Table B.9: Summary of one-sided Wilcoxon rank sum test with significance level $\alpha = 0.01$ for the 100-dimensional benchmarks. For each algorithmic combination (A, B), this matrix shows how often A performed significantly better than B.

100D	1	2	3	4	5	6	7
Hyperbolic (1)	0	12	11	10	9	18	12
RandomBack (2)	3	0	4	4	1	18	4
Nearest-Z (3)	3	5	0	5	0	18	5
Random-Z (4)	3	7	3	0	2	18	4
Reflect-Z (5)	3	9	8	7	0	18	6
Infinity (6)	0	0	0	0	0	0	0
Infinity-C (7)	2	7	4	2	4	18	0

Table B.10: Summary of one-sided Wilcoxon rank sum test with significance level $\alpha = 0.01$ for the 500-dimensional benchmarks. For each algorithmic combination (A, B), this matrix shows how often A performed significantly better than B.

500D	1	2	3	4	5	6	7
Hyperbolic (1)	0	15	15	16	14	18	18
RandomBack (2)	3	0	8	6	2	18	18
Nearest-Z (3)	2	4	0	5	1	18	18
Random-Z (4)	2	10	11	0	7	18	18
Reflect-Z (5)	3	11	13	8	0	18	18
Infinity (6)	0	0	0	0	0	0	0
Infinity-C (7)	0	0	0	0	0	1	0

Table B.11: Summary of one-sided Wilcoxon rank sum test with significance level $\alpha = 0.01$ for the 500-dimensional benchmarks. For each algorithmic combination (A, B), this matrix shows on which benchmarks A performed significantly better than B. $\mathcal{B}$ denotes the set of all benchmark functions.

500D	1	2	3	4	5	6	7
Hyperbolic (1)	{}	{Sph, Ro, Ack, Grie, Ra, f1, f2, f3, f5, f6, f8, f11, f12, f13, f14}	{Sph, Ro, Ack, Grie, Ra, f1, f2, f3, f5, f6, f8, f11, f12, f13, f14}	{Sph, Ro, Ack, Grie, Ra, Schw, f1, f2, f3, f5, f6, f8, f11, f12, f13, f14}	{Sph, Ro, Ack, Grie, Ra, f1, f2, f3, f5, f6, f11, f12, f13, f14}	$\mathcal{B}$	$\mathcal{B}$
Random Back (2)	{Schw, f9, f10}	{}	{Grie, Ra, f1, f2, f3, f6, f12, f13}	{Schw, f1, f2, f3, f6, f12}	{Ack, f12}	$\mathcal{B}$	$\mathcal{B}$
Nearest-Z (3)	{Schw, f10}	{Ro, f10, f11, f14}	{}	{Schw, f1, f3, f10, f11}	{Ack}	$\mathcal{B}$	$\mathcal{B}$
Random-Z (4)	{f9, f10}	{Sph, Ro, Ack, Grie, Ra, f5, f9, f10, f13, f14}	{Sph, Ro, Ack, Grie, Ra, f2, f5, f9, f12, f13, f14}	{}	{Sph, Ro, Ack, Grie, Ra, f5, f13}	$\mathcal{B}$	$\mathcal{B}$
Reflect-Z (5)	{Schw, f9, f10}	{Schw, f1, f2, f3, f5, f6, f9, f10, f11, f13, f14}	{Ra, Schw, f1, f2, f3, f5, f6, f9, f10, f11, f12, f13, f14}	{Schw, f1, f2, f3, f6, f10, f11, f12}	{}	$\mathcal{B}$	$\mathcal{B}$
Infinity (6)	{}	{}	{}	{}	{}	{}	{}
Infinity-C (7)	{}	{}	{}	{}	{}	{f9}	{}

Sample means

Table B.12: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies on various two-dimensional benchmarks. The best objective values are presented together with the function name.

	Sphere (0)	Rosenbrock (0)
Hyperbolic	3.6032e-12±6.7254e-13 (3.3895e-12)	5.2219e-11±1.0313e-11 (5.1974e-11)
RandomBack	**3.4145e-12±5.4319e-13 (2.7376e-12)**	5.5215e-11±1.2517e-11 (6.3084e-11)
Nearest-Z	4.1256e-12±7.4708e-13 (3.7651e-12)	5.4738e-11±1.0448e-11 (5.2657e-11)
Random-Z	4.1156e-12±7.1352e-13 (3.596e-12)	*6.2471e-11±1.3411e-11 (6.7586e-11)*
Reflect-Z	3.9078e-12±8.0818e-13 (4.0731e-12)	5.1719e-11±1.003e-11 (5.0546e-11)
Infinity	*4.3293e-12±9.4978e-13 (4.7867e-12)*	5.5446e-11±1.1616e-11 (5.8544e-11)
Infinity-C	3.7252e-12±7.8633e-13 (3.9629e-12)	**4.4653e-11±9.7937e-12 (4.9358e-11)**
	Ackley (0)	Griewank (0)
Hyperbolic	*1.1223e-08±1.0628e-09 (5.3564e-09)*	4.2826e-12±8.6147e-13 (4.3416e-12)
RandomBack	1.0228e-08±1.0667e-09 (5.376e-09)	4.4019e-12±7.988e-13 (4.0258e-12)
Nearest-Z	**9.3314e-09±1.147e-09 (5.7804e-09)**	*7.396e-05±1.4675e-04 (7.396e-04)*
Random-Z	9.6992e-09±9.1523e-10 (4.6125e-09)	4.368e-12±8.027e-13 (4.0454e-12)
Reflect-Z	9.9458e-09±1.0864e-09 (5.4753e-09)	4.8228e-12±1.1069e-12 (5.5787e-12)
Infinity	9.341e-09±8.8613e-10 (4.4659e-09)	**3.7952e-12±7.3899e-13 (3.7243e-12)**
Infinity-C	9.5409e-09±9.4472e-10 (4.7612e-09)	5.1992e-12±1.0932e-12 (5.5096e-12)
	Rastrigin (0)	Schwefel (≈-837.96)
Hyperbolic	4.3418e-12±7.0524e-13 (3.5543e-12)	*-820.2±8.4338 (42.504)*
RandomBack	*4.4243e-12±9.3472e-13 (4.7108e-12)*	**-837.97 (0)**
Nearest-Z	**3.6217e-12±8.1765e-13 (4.1207e-12)**	**-837.97 (0)**
Random-Z	4.2274e-12±8.0098e-13 (4.0367e-12)	**-837.97 (0)**
Reflect-Z	3.7792e-12±7.0685e-13 (3.5623e-12)	**-837.97 (0)**
Infinity	4.2269e-12±9.0662e-13 (4.5691e-12)	**-837.97 (0)**
Infinity-C	3.6425e-12±7.352e-13 (3.7052e-12)	-836.78±2.3501 (11.844)
	f1 (-450)	f2 (-450)
All strategies	**-450 (0)**	**-450 (0)**

Table B.13: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies on various two-dimensional benchmarks. The best objective values are presented together with the function name.

	f3 (-450)	f5 (-310)
Hyperbolic	**-450 (0)**	**-310 (0)**
RandomBack	**-450 (0)**	**-310 (0)**
Nearest-Z	**-450 (0)**	**-310 (0)**
Random-Z	**-450 (0)**	*-309.13±0.10605 (0.53447)*
Reflect-Z	**-450 (0)**	**-310 (0)**
Infinity	**-450 (0)**	**-310 (0)**
Infinity-C	**-450 (0)**	**-310 (0)**
	f6 (390)	f8 (-140)
Hyperbolic	**390 (0)**	*-122.12±1.0591 (5.3374)*
RandomBack	**390 (0)**	**-140 (0)**
Nearest-Z	**390 (0)**	-139.8±0.39684 (2)
Random-Z	**390 (0)**	-139.75±0.39933 (2.0125)
Reflect-Z	**390 (0)**	-139.97±0.051193 (0.258)
Infinity	**390 (0)**	-139.55±0.56155 (2.8301)
Infinity-C	**390 (0)**	-138.75±0.94732 (4.7743)
	f9 (-330)	f10 (-330)
All strategies	**-330 (0)**	**-330 (0)**
	f11 (-460)	f12 (90)
Hyperbolic	**90±1.1087e-05 (5.5877e-05)**	**-460 (0)**
RandomBack	**90±1.213e-05 (6.1134e-05)**	**-460 (0)**
Nearest-Z	**90±1.3401e-05 (6.754e-05)**	**-460 (0)**
Random-Z	**90±1.1647e-05 (5.8698e-05)**	**-460 (0)**
Reflect-Z	**90±1.1323e-05 (5.7066e-05)**	**-460 (0)**
Infinity	**90±1.2287e-05 (6.1922e-05)**	**-460 (0)**
Infinity-C	**90±1.4107e-05 (7.1095e-05)**	**-460 (0)**
	f13 (-130)	f14 (-300)
Hyperbolic	**-130±3.9684e-04 (2e-03)**	**-300±3.77e-04 (1.9e-03)**
RandomBack	**-130 (0)**	**-300 (0)**
Nearest-Z	**-130±3.9684e-04 (2e-03)**	**-300 (0)**
Random-Z	**-130±3.9684e-04 (2e-03)**	**-300 (0)**
Reflect-Z	**-130 (0)**	**-300 (0)**
Infinity	**-130 (0)**	**-300 (0)**
Infinity-C	**-130 (0)**	**-300 (0)**

Table B.14: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies on various 30-dimensional benchmarks. The best objective values are presented together with the function name.

	Sphere (0)	Rosenbrock (0)
Hyperbolic	9.5661e-07±2.5178e-08 (1.2689e-07)	18.487±3.3325 (16.795)
RandomBack	9.5149e-07±3.2691e-08 (1.6475e-07)	18.252±4.1707 (21.019)
Nearest-Z	**9.4625e-07±2.7552e-08 (1.3886e-07)**	*20.643±4.7629 (24.004)*
Random-Z	9.5121e-07±2.8048e-08 (1.4135e-07)	**17.338±3.4392 (17.333)**
Reflect-Z	*9.6761e-07±2.6412e-08 (1.3311e-07)*	17.87±4.0681 (20.502)
Infinity	9.6255e-07±2.9661e-08 (1.4949e-07)	19.013±4.2857 (21.599)
Infinity-C	9.6111e-07±2.8789e-08 (1.4509e-07)	17.471±4.0357 (20.339)
	Ackley (0)	Griewank (0)
Hyperbolic	2.7718e-06±5.5279e-08 (2.7859e-07)	3.9159e-03±1.3217e-03 (6.6611e-03)
RandomBack	2.8583e-06±5.1333e-08 (2.5871e-07)	5.8621e-03±1.7887e-03 (9.0146e-03)
Nearest-Z	**2.7581e-06±6.0215e-08 (3.0347e-07)**	5.3402e-03±1.8586e-03 (9.367e-03)
Random-Z	*0.013407±0.026597 (0.13404)*	4.0904e-03±1.3164e-03 (6.6344e-03)
Reflect-Z	2.7915e-06±5.5928e-08 (2.8186e-07)	6.4001e-03±1.8506e-03 (9.3266e-03)
Infinity	2.8183e-06±5.4269e-08 (2.735e-07)	**3.3264e-03±1.2135e-03 (6.1156e-03)**
Infinity-C	2.8375e-06±5.2596e-08 (2.6507e-07)	*6.5502e-03±1.7536e-03 (8.8377e-03)*
	Rastrigin (0)	Schwefel (≈-837.96)
Hyperbolic	**28.874±1.4589 (7.3523)**	*-8049.1±133.06 (670.6)*
RandomBack	42.684±1.9667 (9.9118)	-9309.2±112.25 (565.73)
Nearest-Z	51.549±2.7688 (13.954)	-9624.3±108.58 (547.22)
Random-Z	40.783±1.9356 (9.7552)	-8903.3±110.5 (556.91)
Reflect-Z	*52.474±2.7422 (13.82)*	**-10470±113.97 (574.41)**
Infinity	49.529±2.4716 (12.456)	-8463.9±109.7 (552.88)
Infinity-C	38.973±2.0172 (10.166)	-8698.5±123.17 (620.74)
	f1 (-450)	f2 (-450)
Hyperbolic	**-450 (0)**	**-450 (0)**
RandomBack	**-450 (0)**	**-450±4.736e-05 (2.3868e-04)**
Nearest-Z	**-450 (0)**	*-429.83±24.755 (124.76)*
Random-Z	**-450 (0)**	**-450±4.3463e-05 (2.1904e-04)**
Reflect-Z	**-450 (0)**	**-450±1.9842e-05 (1e-04)**
Infinity	**-450 (0)**	**-450 (0)**
Infinity-C	**-450 (0)**	**-450 (0)**

Table B.15: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies on various 30-dimensional benchmarks. The best objective values are presented together with the function name.

	f3 (-450)	f5 (-310)
Hyperbolic	**738070±58951 (297100)**	3804.7±121.26 (611.12)
RandomBack	1475200±130650 (658440)	3724.6±180.71 (910.72)
Nearest-Z	*2316000±208220 (1049400)*	3547.8±201.09 (1013.4)
Random-Z	1569200±198900 (1002400)	4048.3±138.95 (700.27)
Reflect-Z	1442700±130660 (658510)	**3538.4±165.24 (832.76)**
Infinity	873640±67303 (339190)	*4479.4±202.63 (1021.2)*
Infinity-C	874710±76215 (384110)	3888.6±159.06 (801.64)
	f6 (390)	f8 (-140)
Hyperbolic	*446.95±9.4215 (47.482)*	**-119.14±0.01257 (0.063348)**
RandomBack	440.67±13.633 (68.708)	*-119.13±0.013896 (0.070034)*
Nearest-Z	444.91±15.337 (77.296)	*-119.13±0.011557 (0.058244)*
Random-Z	432.53±10.318 (51.999)	*-119.13±0.01084 (0.054631)*
Reflect-Z	429.34±11.857 (59.756)	*-119.13±0.01082 (0.05453)*
Infinity	430.07±9.4938 (47.847)	*-119.13±0.012033 (0.060644)*
Infinity-C	**424.57±9.0452 (45.586)**	*-119.13±0.014156 (0.071342)*
	f9 (-330)	f10 (-330)
Hyperbolic	*-218.59±3.5826 (18.056)*	*-206.73±4.4929 (22.643)*
RandomBack	-281.37±2.3642 (11.915)	-263.84±3.1699 (15.976)
Nearest-Z	-279.65±2.3164 (11.674)	-264.88±3.2648 (16.454)
Random-Z	-281.73±2.3685 (11.937)	-266.51±3.2003 (16.129)
Reflect-Z	**-283.79±2.2435 (11.307)**	**-274.63±2.5824 (13.015)**
Infinity	-259.38±4.0219 (20.27)	-228.17±5.4054 (27.242)
Infinity-C	-267.95±3.2483 (16.37)	-254.76±3.6327 (18.308)
	f11 (-460)	f12 (90)
Hyperbolic	**115.06±0.67756 (3.4148)**	4888.2±1211.1 (6103.5)
RandomBack	118.26±0.39537 (1.9926)	11885±2352.3 (11855)
Nearest-Z	116.98±0.54618 (2.7526)	*20912±4247.1 (21405)*
Random-Z	116.65±0.52443 (2.643)	**2696.7±580.43 (2925.2)**
Reflect-Z	116.58±0.58174 (2.9318)	11863±2629.7 (13253)
Infinity	*118.33±0.5223 (2.6323)*	7535±1596.1 (8044)
Infinity-C	116.66±0.57712 (2.9085)	5708.9±1198.9 (6042.2)
	f13 (-130)	f14 (-300)
Hyperbolic	**-126.96±0.13978 (0.70445)**	-287.91±0.080686 (0.40664)
RandomBack	-125.5±0.2305 (1.1617)	-287.74±0.086898 (0.43795)
Nearest-Z	-125.72±0.23507 (1.1847)	-287.71±0.071834 (0.36203)
Random-Z	-125.79±0.20734 (1.0449)	**-287.95±0.085788 (0.43235)**
Reflect-Z	-125.88±0.22082 (1.1129)	-287.87±0.073504 (0.37045)
Infinity	*-125.11±0.29755 (1.4996)*	*-287.64±0.080278 (0.40458)*
Infinity-C	-125.76±0.22892 (1.1537)	-287.92±0.086136 (0.43411)

Table B.16: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies on various 100-dimensional benchmarks. The best objective values are presented together with the function name.

	Sphere (0)	Rosenbrock (0)
Hyperbolic	**5.9506e-06±1.6691e-07 (8.412e-07)**	**152.94±9.4227 (47.488)**
RandomBack	6.1517e-06±1.9396e-07 (9.775e-07)	203.93±12.355 (62.268)
Nearest-Z	5.9515e-06±1.6588e-07 (8.3601e-07)	200.52±15.094 (76.069)
Random-Z	6.132e-06±1.6757e-07 (8.445e-07)	201.9±9.2629 (46.683)
Reflect-Z	6.0293e-06±1.7838e-07 (8.9898e-07)	176.34±10.406 (52.445)
Infinity	*11006±5948.2 (29978)*	*46114000±20048000 (1.0104e+08)*
Infinity-C	6.0565e-06±1.7488e-07 (8.8135e-07)	192.27±9.6391 (48.579)
	Ackley (0)	Griewank (0)
Hyperbolic	**0.37429±0.12799 (0.64502)**	**3.5747e-03±1.3876e-03 (6.9929e-03)**
RandomBack	1.6707±0.15722 (0.79235)	7.5186e-03±3.5184e-03 (0.017732)
Nearest-Z	1.667±0.12876 (0.64891)	4.5718e-03±2.1514e-03 (0.010842)
Random-Z	1.5611±0.14517 (0.73164)	3.7379e-03±1.9864e-03 (0.010011)
Reflect-Z	1.6946±0.13572 (0.68397)	3.8688e-03±1.6439e-03 (8.2849e-03)
Infinity	*17.622±0.28504 (1.4365)*	*125.88±58.171 (293.17)*
Infinity-C	1.751±0.12394 (0.62462)	6.1172e-03±3.6482e-03 (0.018386)
	Rastrigin (0)	Schwefel (≈-837.96)
Hyperbolic	**105.46±4.1927 (21.13)**	-24848±333.63 (1681.4)
RandomBack	337.09±8.0286 (40.462)	-27196±270.12 (1361.3)
Nearest-Z	366.22±7.7675 (39.146)	-28437±309.36 (1559.1)
Random-Z	259.75±6.483 (32.673)	-25510±232.94 (1174)
Reflect-Z	355.61±7.7081 (38.847)	**-30569±305.32 (1538.7)**
Infinity	*745.71±58.688 (295.77)*	*-6643.8±353.75 (1782.8)*
Infinity-C	262.51±6.5867 (33.195)	-23686±352.02 (1774.1)
	f1 (-450)	f2 (-450)
Hyperbolic	**-450 (0)**	**3348.3±123.56 (622.7)**
RandomBack	**-450 (0)**	28483±1480.5 (7461.5)
Nearest-Z	-447.36±5.2478 (26.447)	41490±2436.5 (12279)
Random-Z	**-450 (0)**	25170±1732.2 (8729.9)
Reflect-Z	**-450 (0)**	23506±1217.5 (6135.7)
Infinity	*188740±20674 (104190)*	*969000±90586 (456540)*
Infinity-C	**-450 (0)**	10348±649.64 (3274)

Table B.17: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies on various 100-dimensional benchmarks. The best objective values are presented together with the function name.

	f3 (-450)	f5 (-310)
Hyperbolic	**12062000±446790 (2251700)**	27956±572.45 (2885)
RandomBack	34425000±1673800 (8435400)	30138±851.15 (4289.6)
Nearest-Z	52624000±3151500 (15883000)	29180±765.47 (3857.8)
Random-Z	7.557e+07±5833300 (29399000)	31841±675.46 (3404.2)
Reflect-Z	3.613e+07±1841200 (9279000)	**27456±697.98 (3517.7)**
Infinity	*8.3934e+09±8.8535e+08 (4.462e+09)*	*93483±1843.9 (9292.9)*
Infinity-C	18045000±986020 (4969300)	33556±836.74 (4217)
	f6 (390)	f8 (-140)
Hyperbolic	**570.68±8.9378 (45.045)**	**-118.71±6.4444e-03 (0.032478)**
RandomBack	594.18±11.698 (58.955)	**-118.71±5.8902e-03 (0.029685)**
Nearest-Z	1347200±2624700 (13228000)	**-118.71±5.9884e-03 (0.03018)**
Random-Z	587.45±10.702 (53.936)	-118.7±5.5891e-03 (0.028168)
Reflect-Z	583.66±10.556 (53.201)	**-118.71±5.8188e-03 (0.029325)**
Infinity	*1.0592e+11±1.558e+10 (7.8521e+10)*	*-118.61±8.2597e-03 (0.041627)*
Infinity-C	580.15±10.005 (50.425)	-118.7±6.6952e-03 (0.033742)
	f9 (-330)	f10 (-330)
Hyperbolic	264.95±9.7299 (49.036)	478.7±15.556 (78.397)
RandomBack	37.5±11.473 (57.821)	104.24±12.998 (65.508)
Nearest-Z	32.932±10.413 (52.481)	49.191±12.657 (63.786)
Random-Z	59.29±10.966 (55.267)	102.2±14.465 (72.899)
Reflect-Z	**3.579±10.44 (52.615)**	**8.2564±12.608 (63.541)**
Infinity	*1029.3±68.735 (346.41)*	*1904.8±73.076 (368.28)*
Infinity-C	236.02±14.843 (74.804)	285.34±21.161 (106.65)
	f11 (-460)	f12 (90)
Hyperbolic	**214.65±1.5314 (7.7177)**	**173200±15811 (79686)**
RandomBack	224.97±1.0398 (5.2402)	554190±45331 (228460)
Nearest-Z	222.97±1.081 (5.4479)	1135300±124970 (629840)
Random-Z	222.81±1.2242 (6.1699)	356330±32520 (163900)
Reflect-Z	218.24±1.2826 (6.4642)	599230±54540 (274870)
Infinity	*258.07±1.138 (5.7355)*	*32666000±2.04e+06 (10281000)*
Infinity-C	222.24±1.3244 (6.6746)	331750±31094 (156710)
	f13 (-130)	f14 (-300)
Hyperbolic	**-102.77±0.93404 (4.7074)**	**-254.18±0.11919 (0.60069)**
RandomBack	-59.652±2.9796 (15.017)	-253.18±0.072314 (0.36445)
Nearest-Z	-65.684±2.6176 (13.192)	-253.41±0.090291 (0.45504)
Random-Z	-64.966±2.9668 (14.952)	-253.43±0.090736 (0.45729)
Reflect-Z	-65.84±2.7268 (13.742)	-253.44±0.079587 (0.4011)
Infinity	*676650±209660 (1056600)*	*-251.65±0.10842 (0.5464)*
Infinity-C	-66.296±2.834 (14.282)	-253.49±0.088254 (0.44478)

Table B.18: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies on various 500-dimensional benchmarks. The best objective values are presented together with the function name.

	Sphere (0)	Rosenbrock (0)
Hyperbolic	**7.4707±0.99067 (4.9928)**	**2791.9±49.433 (249.13)**
RandomBack	2671.1±319.63 (1610.9)	1436500±203620 (1026200)
Nearest-Z	3598.9±648.65 (3269.1)	1986100±1605300 (8090300)
Random-Z	1725.5±178.35 (898.82)	966960±141380 (712510)
Reflect-Z	2778.6±294.31 (1483.2)	1619400±278580 (1404000)
Infinity	*786330±4787.4 (24127)*	*2.3542e+09±23117000 (1.165e+08)*
Infinity-C	775590±10092 (50861)	2.3196e+09±38858000 (1.9584e+08)
	Ackley (0)	Griewank (0)
Hyperbolic	**3.3085±0.058416 (0.2944)**	**0.58759±0.04207 (0.21202)**
RandomBack	12.676±0.3083 (1.5538)	22.369±2.156 (10.866)
Nearest-Z	12.887±0.35175 (1.7728)	34.214±6.3247 (31.875)
Random-Z	10.415±0.2128 (1.0724)	17.373±1.7809 (8.9753)
Reflect-Z	13.785±0.30933 (1.5589)	27.222±3.6796 (18.545)
Infinity	*20.075±0.013239 (0.066721)*	*7042.8±42.91 (216.26)*
Infinity-C	*20.075±0.012541 (0.063205)*	6954±90.637 (456.79)
	Rastrigin (0)	Schwefel (≈-837.96)
Hyperbolic	**467.81±12.732 (64.167)**	-116190±1305.6 (6579.7)
RandomBack	2360.3±40.692 (205.08)	-122750±1000.3 (5041.3)
Nearest-Z	2435.5±33.708 (169.88)	-121790±1250.9 (6304.3)
Random-Z	1804.2±34.68 (174.78)	-109020±986.27 (4970.6)
Reflect-Z	2341.1±31.406 (158.28)	**-138250±1312.1 (6612.6)**
Infinity	*6727.1±20.394 (102.78)*	-11363±370.62 (1867.8)
Infinity-C	6725.7±20.941 (105.54)	*-11309±346.66 (1747.1)*
	f1 (-450)	f2 (-450)
Hyperbolic	**-417.69±2.2298 (11.238)**	**954060±12772 (64369)**
RandomBack	3350.5±481.59 (2427.1)	2460100±38217 (192600)
Nearest-Z	7009.6±938.12 (4727.9)	2719900±57572 (290150)
Random-Z	23251±1449.8 (7306.5)	2604300±56580 (285150)
Reflect-Z	2269.2±349.6 (1761.9)	2305700±42498 (214180)
Infinity	*2184100±11527 (58096)*	98947000±6771500 (34127000)
Infinity-C	2174800±13203 (66538)	*1.0007e+08±8033800 (40489000)*

Table B.19: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies on various 500-dimensional benchmarks. The best objective values are presented together with the function name.

	f3 (-450)	f5 (-310)
Hyperbolic	**4.0784e+08±5140200 (25906000)**	**171200±942.14 (4748.2)**
RandomBack	1.9081e+09±47268000 (2.3822e+08)	195600±1621.7 (8173.1)
Nearest-Z	2.105e+09±56322000 (2.8385e+08)	195240±1743.8 (8788.1)
Random-Z	3.4002e+09±1.3766e+08 (6.9378e+08)	182970±1331 (6707.9)
Reflect-Z	1.7836e+09±54176000 (2.7304e+08)	188720±1735.5 (8746.3)
Infinity	*1.1247e+11±1.622e+09 (8.1745e+09)*	*283680±1824.6 (9195.7)*
Infinity-C	1.1126e+11±1.6522e+09 (8.3265e+09)	280520±1827.7 (9211)
	f6 (390)	f8 (-140)
Hyperbolic	**49565±4214.7 (21241)**	**-118.46±2.4297e-03 (0.012245)**
RandomBack	5.5106e+08±1.1595e+08 (5.8437e+08)	**-118.46±2.1463e-03 (0.010817)**
Nearest-Z	2.4123e+09±6.21e+08 (3.1297e+09)	**-118.46±2.0048e-03 (0.010104)**
Random-Z	1.3169e+09±1.9302e+08 (9.7276e+08)	-118.45±2.3646e-03 (0.011917)
Reflect-Z	4.001e+08±1.0903e+08 (5.4947e+08)	**-118.46±2.5188e-03 (0.012694)**
Infinity	*2.0513e+12±2.1818e+10 (1.0996e+11)*	*-118.41±3.1054e-03 (0.01565)*
Infinity-C	2.0318e+12±2.1513e+10 (1.0842e+11)	-118.42±2.9722e-03 (0.014979)
	f9 (-330)	f10 (-330)
Hyperbolic	3695.4±28.08 (141.52)	7320±61.405 (309.46)
RandomBack	3562.4±62.864 (316.82)	5940.5±122.51 (617.44)
Nearest-Z	3643.8±61.275 (308.81)	5043.5±99.932 (503.63)
Random-Z	**3153.4±53.629 (270.28)**	5684.7±103.05 (519.33)
Reflect-Z	3189.1±69.075 (348.12)	**4460.7±112.76 (568.28)**
Infinity	*10265±33.796 (170.32)*	*16910±69.904 (352.3)*
Infinity-C	10170±39.467 (198.9)	16871±74.97 (377.83)
	f11 (-460)	f12 (90)
Hyperbolic	**892.79±4.5699 (23.031)**	**18562000±918840 (4630800)**
RandomBack	926.43±2.5768 (12.987)	53336000±1625500 (8191900)
Nearest-Z	917.43±2.783 (14.026)	1.0481e+08±5946900 (29971000)
Random-Z	926.3±2.3777 (11.983)	81782000±2962900 (14932000)
Reflect-Z	910.36±2.7508 (13.863)	61999000±2113500 (10652000)
Infinity	*1029.4±1.8808 (9.4789)*	1.3203e+09±9006400 (4.539e+07)
Infinity-C	1029.1±1.9532 (9.8437)	*1.3254e+09±8284700 (41753000)*
	f13 (-130)	f14 (-300)
Hyperbolic	**602.78±15.087 (76.033)**	**-56.356±0.19687 (0.9922)**
RandomBack	21004±2593.7 (13072)	-54.331±0.08153 (0.41089)
Nearest-Z	91208±12353 (62257)	-54.547±0.082538 (0.41597)
Random-Z	5505.3±958.1 (4828.6)	-54.717±0.078315 (0.39469)
Reflect-Z	10424±1558.2 (7852.8)	-54.743±0.095998 (0.48381)
Infinity	*46733000±1238600 (6242200)*	*-52.668±0.08062 (0.40631)*
Infinity-C	46558000±1250300 (6301000)	-52.784±0.085992 (0.43338)

B.4 Velocity Handling

Detailed results for the experimental comparison of different velocity handling strategies (Section 4.4.3):

- Summary of Wilcoxon rank sum test: Table B.20
- Sample means, 95% confidence intervals, sample standard deviations of $N = 100$ runs: Tables B.21, B.22, and B.23

Table B.20: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows on which benchmarks A performed significantly better than B. Rosenbrock, Rastrigin, and Schwefel were abbreviated.

	1	2	3
Nearest-Z (1)	{}	{Schw, f1–f6, f9, f12}	{Sphere, Ra, Schw, f1–f6, f9, f10, f12}
Nearest-A (2)	{}	{}	{Ra, Schw, f1–f6, f9, f10, f12}
Nearest-U (3)	{}	{}	{}
	4	5	6
Random-Z (4)	{}	{Schw, f2, f3, f5, f12}	{Schw, f1–f12, f14}
Random-A (5)	{Ro, Ra}	{}	{Schw, f1–f12, f14}
Random-U (6)	{Ra}	{Ra}	{}
	7	8	9
Reflect-Z (7)	{}	{Ro}	{Schw, f1–f12, f14}
Reflect-A (8)	{}	{}	{Schw, f1–f12, f14}
Reflect-U (9)	{Ra}	{Ra}	{}

Table B.21: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of different bound handling strategies on 100-dimensional benchmarks. The best objective values are given with the function name.

	Sphere (0)	Rosenbrock (0)
Nearest-Z	**5.9515e-06±1.6588e-07 (8.3601e-07)**	**200.52±15.094 (76.069)**
Nearest-A	6.1284e-06±1.6965e-07 (8.5499e-07)	1134±1786.8 (9005.3)
Nearest-U	*100±198.42 (1e+03)*	*3950.6±3511.6 (17697)*
	Ackley (0)	Griewank (0)
Nearest-Z	1.667±0.12876 (0.64891)	*4.5718e-03±2.1514e-03 (0.010842)*
Nearest-A	*1.6681±0.13927 (0.70187)*	4.5319e-03±1.7501e-03 (8.8201e-03)
Nearest-U	**1.6332±0.14159 (0.71357)**	**3.0603e-03±1.1242e-03 (5.6658e-03)**
	Rastrigin (0)	Schwefel (≈ -41898.3)
Nearest-Z	**366.22±7.7675 (39.146)**	**-28437±309.36 (1559.1)**
Nearest-A	374.9±8.3297 (41.98)	-27707±320.35 (1614.5)
Nearest-U	*398.33±8.1286 (40.966)*	*-25241±378.91 (1909.6)*
	f1 (-450)	f2 (-450)
Nearest-Z	**-447.36±5.2478 (26.447)**	**41490±2436.5 (12279)**
Nearest-A	-378.33±45.357 (228.59)	60542±5034.7 (25374)
Nearest-U	*2828.6±623.93 (3144.5)*	*84154±6604.3 (33284)*
	f3 (-450)	f5 (-310)
Nearest-Z	**52624000±3151500 (15883000)**	**29180±765.47 (3857.8)**
Nearest-A	85561000±8118300 (40914000)	31552±998.58 (5032.6)
Nearest-U	*2.1674e+08±34632000 (1.7454e+08)*	*34212±1132.1 (5705.6)*
	f6 (390)	f8 (-140)
Nearest-Z	**1347200±2624700 (13228000)**	**-118.71±5.9884e-03 (0.03018)**
Nearest-A	2.246e+07±16277000 (82031000)	*-118.7±5.798e-03 (0.029221)*
Nearest-U	*5.4961e+08±1.6873e+08 (8.5038e+08)*	**-118.71±5.4093e-03 (0.027262)**
	f9 (-330)	f10 (-330)
Nearest-Z	**32.932±10.413 (52.481)**	**49.191±12.657 (63.786)**
Nearest-A	59.344±11.94 (60.177)	59.421±11.883 (59.885)
Nearest-U	*105.91±12.678 (63.895)*	*89.804±13.731 (69.201)*
	f11 (-460)	f12 (90)
Nearest-Z	222.97±1.081 (5.4479)	**1135300±124970 (629840)**
Nearest-A	**221.77±1.2211 (6.154)**	1904300±166250 (837850)
Nearest-U	*223.29±1.0891 (5.489)*	*3912500±299620 (1.51e+06)*
	f13 (-130)	f14 (-300)
Nearest-Z	**-65.684±2.6176 (13.192)**	-253.41±0.090291 (0.45504)
Nearest-A	37.248±202.33 (1019.7)	**-253.43±0.10147 (0.5114)**
Nearest-U	*1962.3±1536.9 (7745.7)*	*-253.4±0.080854 (0.40749)*

Table B.22: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of different bound handling strategies on 100-dimensional benchmarks. The best objective values are given with the function name.

	Sphere (0)	Rosenbrock (0)
Random-Z	*6.132e-06±1.6757e-07 (8.445e-07)*	*201.9±9.2629 (46.683)*
Random-A	**6.0663e-06±1.8517e-07 (9.332e-07)**	**185.37±9.6231 (48.498)**
Random-U	6.0751e-06±1.532e-07 (7.7208e-07)	192.36±10.844 (54.653)
	Ackley (0)	Griewank (0)
Random-Z	**1.5611±0.14517 (0.73164)**	**3.7379e-03±1.9864e-03 (0.010011)**
Random-A	1.6038±0.16066 (0.80969)	*6.714e-03±2.2943e-03 (0.011563)*
Random-U	*1.6486±0.15815 (0.79706)*	6.4136e-03±3.314e-03 (0.016702)
	Rastrigin (0)	Schwefel (≈ -41898.3)
Random-Z	*259.75±6.483 (32.673)*	**-25510±232.94 (1174)**
Random-A	235.82±5.3966 (27.198)	-23989±267.19 (1346.6)
Random-U	**169.25±4.2255 (21.295)**	*-13598±213.13 (1074.1)*
	f1 (-450)	f2 (-450)
Random-Z	**-450 (0)**	**25170±1732.2 (8729.9)**
Random-A	**-450 (0)**	37500±3348.3 (16875)
Random-U	*45968±1277.4 (6437.8)*	*129710±5761.8 (29038)*
	f3 (-450)	f5 (-310)
Random-Z	**7.557e+07±5833300 (29399000)**	**31841±675.46 (3404.2)**
Random-A	1.508e+08±14568000 (73422000)	34743±713.53 (3596)
Random-U	*7.5821e+08±45569000 (2.2966e+08)*	*52870±579.7 (2921.6)*
	f6 (390)	f8 (-140)
Random-Z	**587.45±10.702 (53.936)**	**-118.7±5.5891e-03 (0.028168)**
Random-A	590.4±11.441 (57.659)	**-118.7±6.4083e-03 (0.032296)**
Random-U	*3.6181e+09±1.5377e+08 (7.7494e+08)*	*-118.66±4.7472e-03 (0.023925)*
	f9 (-330)	f10 (-330)
Random-Z	**59.29±10.966 (55.267)**	**102.2±14.465 (72.899)**
Random-A	75.882±10.652 (53.684)	127.88±15.427 (77.751)
Random-U	*180.07±14.437 (72.757)*	*489.68±34.621 (174.48)*
	f11 (-460)	f12 (90)
Random-Z	**222.81±1.2242 (6.1699)**	**356330±32520 (163900)**
Random-A	223.59±1.012 (5.1003)	781250±79678 (401560)
Random-U	*240.24±1.1665 (5.879)*	*6499000±228620 (1152200)*
	f13 (-130)	f14 (-300)
Random-Z	-64.966±2.9668 (14.952)	**-253.43±0.090736 (0.45729)**
Random-A	**-66.017±2.6414 (13.312)**	-253.39±0.074503 (0.37548)
Random-U	*-64.674±3.0484 (15.363)*	*-252.78±0.079395 (0.40013)*

Table B.23: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of different bound handling strategies on 100-dimensional benchmarks. The best objective values are given with the function name.

	Sphere (0)	Rosenbrock (0)
Reflect-Z	**6.0293e-06±1.7838e-07 (8.9898e-07)**	**176.34±10.406 (52.445)**
Reflect-A	*6.2182e-06±1.9525e-07 (9.8402e-07)*	*206.71±12.274 (61.857)*
Reflect-U	6.1753e-06±1.59e-07 (8.0132e-07)	186.37±10.486 (52.846)
	Ackley (0)	Griewank (0)
Reflect-Z	1.6946±0.13572 (0.68397)	**3.8688e-03±1.6439e-03 (8.2849e-03)**
Reflect-A	**1.6295±0.14767 (0.74424)**	3.9465e-03±1.3659e-03 (6.8836e-03)
Reflect-U	*1.7091±0.13779 (0.69443)*	*4.8019e-03±2.8064e-03 (0.014144)*
	Rastrigin (0)	Schwefel (≈ -41898.3)
Reflect-Z	355.61±7.7081 (38.847)	**-30569±305.32 (1538.7)**
Reflect-A	*361.01±8.8748 (44.727)*	-30208±300.56 (1514.8)
Reflect-U	**245.79±6.2749 (31.624)**	*-26245±308.4 (1554.2)*
	f1 (-450)	f2 (-450)
Reflect-Z	**-450 (0)**	**23506±1217.5 (6135.7)**
Reflect-A	**-450 (0)**	25744±1365.3 (6880.9)
Reflect-U	*12766±952.26 (4799.2)*	*130380±8331.4 (41988)*
	f3 (-450)	f5 (-310)
Reflect-Z	**3.613e+07±1841200 (9279000)**	27456±697.98 (3517.7)
Reflect-A	37901000±1953700 (9846100)	**27141±870.09 (4385.1)**
Reflect-U	*8.4241e+08±45487000 (2.2924e+08)*	*43793±749.61 (3777.8)*
	f6 (390)	f8 (-140)
Reflect-Z	**583.66±10.556 (53.201)**	**-118.71±5.8188e-03 (0.029325)**
Reflect-A	591.76±14.686 (74.016)	**-118.71±6.7394e-03 (0.033965)**
Reflect-U	*6.9439e+08±9.089e+07 (4.5807e+08)*	*-118.66±5.0359e-03 (0.02538)*
	f9 (-330)	f10 (-330)
Reflect-Z	**3.579±10.44 (52.615)**	8.2564±12.608 (63.541)
Reflect-A	3.9619±10.82 (54.528)	**3.606±13.029 (65.662)**
Reflect-U	*61.014±9.7036 (48.904)*	*435.11±31.537 (158.94)*
	f11 (-460)	f12 (90)
Reflect-Z	**218.24±1.2826 (6.4642)**	**599230±54540 (274870)**
Reflect-A	219.38±1.3435 (6.771)	674880±52562 (264900)
Reflect-U	*234.95±1.2996 (6.5495)*	*4452400±275620 (1389000)*
	f13 (-130)	f14 (-300)
Reflect-Z	-65.84±2.7268 (13.742)	-253.44±0.079587 (0.4011)
Reflect-A	**-66.894±3.1004 (15.625)**	**-253.45±0.080678 (0.4066)**
Reflect-U	*-64.258±2.912 (14.676)*	*-252.77±0.06368 (0.32093)*

B.5 Multi-Swarm PSO with Migration

Detailed results for the experimental investigation of *Multi-Swarm PSO with Migration* (see Section 5.2).

Table B.24: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) on various 100-dimensional benchmarks. The best objective values are presented together with the function name.

	Sphere (0)	Rosenbrock (0)
MPSO-1-5	**1.0488e-06±5.5019e-08 (2.7728e-07)**	166.4±10.032 (50.561)
MPSO-1-50	1.1356e-06±6.4714e-08 (3.2614e-07)	**162.69±9.4948 (47.852)**
MPSO-1-100	1.1077e-06±7.5428e-08 (3.8014e-07)	*187.1±9.5036 (47.896)*
MPSO-1-200	1.1528e-06±6.2315e-08 (3.1405e-07)	174.91±9.1728 (46.229)
MPSO-1-500	*1.5955e-06±9.9972e-08 (5.0384e-07)*	163.47±8.0403 (40.521)
	Ackley (0)	Griewank (0)
MPSO-1-5	**0.28922±0.11452 (0.57717)**	**1.7498e-03±9.087e-04 (4.5796e-03)**
MPSO-1-50	1.6472±0.12751 (0.64264)	5.5424e-03±3.3698e-03 (0.016983)
MPSO-1-100	1.7576±0.12814 (0.64582)	*8.5606e-03±4.9031e-03 (0.02471)*
MPSO-1-200	1.9556±0.10138 (0.51094)	5.3296e-03±2.8333e-03 (0.014279)
MPSO-1-500	*2.0487±0.10697 (0.53913)*	1.8724e-03±1.1101e-03 (5.5946e-03)
	Rastrigin (0)	Schwefel (≈ -41898.3)
MPSO-1-5	**115.52±4.9594 (24.994)**	*-28255±560.67 (2825.7)*
MPSO-1-50	121.21±4.8256 (24.32)	-29648±311.78 (1571.3)
MPSO-1-100	123.61±5.3266 (26.845)	-29715±337.31 (1700)
MPSO-1-200	121.15±4.517 (22.765)	**-29768±299.94 (1511.6)**
MPSO-1-500	*124.64±5.0225 (25.312)*	-29725±295.27 (1488.1)
	f1 (-450)	f2 (-450)
MPSO-1-5	**-450 (0)**	**3452.3±125.47 (632.32)**
MPSO-1-50	**-450 (0)**	3634.5±136.6 (688.42)
MPSO-1-100	**-450 (0)**	3689.3±147.04 (741.06)
MPSO-1-200	**-450 (0)**	3750.2±137.55 (693.24)
MPSO-1-500	**-450 (0)**	*3815.5±134.33 (677)*

Table B.25: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) on various 100-dimensional benchmarks. The best objective values are presented together with the function name.

	f3 (-450)	f5 (-310)
MPSO-1-5	**11953000±501610 (2528000)**	**28398±618.69 (3118)**
MPSO-1-50	12494000±554620 (2795200)	30050±687.01 (3462.4)
MPSO-1-100	12706000±446140 (2248400)	29708±673.38 (3393.7)
MPSO-1-200	*1.284e+07±517990 (2610600)*	*30656±651.62 (3284)*
MPSO-1-500	12312000±491630 (2477700)	29771±626.22 (3156)
	f6 (390)	f8 (-140)
MPSO-1-5	**565.95±8.4243 (42.457)**	**-118.71±6.0264e-03 (0.030372)**
MPSO-1-50	573.35±11.013 (55.501)	**-118.71±6.5725e-03 (0.033124)**
MPSO-1-100	567.62±10.023 (50.515)	**-118.71±6.1638e-03 (0.031064)**
MPSO-1-200	*579.25±10.284 (51.829)*	**-118.71±5.2862e-03 (0.026641)**
MPSO-1-500	576.27±9.3356 (47.049)	**-118.71±6.4494e-03 (0.032503)**
	f9 (-330)	f10 (-330)
MPSO-1-5	*220.61±17.76 (89.506)*	*164.61±18.912 (95.311)*
MPSO-1-50	71.914±14.208 (71.604)	**95.904±17.682 (89.112)**
MPSO-1-100	**68.915±13.298 (67.02)**	101.54±17.918 (90.302)
MPSO-1-200	95.825±13.55 (68.288)	101.1±17.228 (86.827)
MPSO-1-500	80.868±15.955 (80.41)	98.86±17.196 (86.662)
	f11 (-460)	f12 (90)
MPSO-1-5	217.32±1.5855 (7.9905)	**198500±21410 (107900)**
MPSO-1-50	216.56±1.5154 (7.6373)	216060±21088 (106280)
MPSO-1-100	*217.79±1.4298 (7.2058)*	*229570±22491 (113350)*
MPSO-1-200	216.69±1.6925 (8.5298)	226640±23815 (120020)
MPSO-1-500	**216.22±1.4888 (7.503)**	207900±18376 (92611)
	f13 (-130)	f14 (-300)
MPSO-1-5	**-96.778±1.3241 (6.6733)**	**-254.08±0.11419 (0.57549)**
MPSO-1-50	-89.361±1.59 (8.0132)	-254.02±0.12191 (0.61441)
MPSO-1-100	-87.593±1.8549 (9.3484)	-253.88±0.10943 (0.55149)
MPSO-1-200	-85.958±1.6669 (8.4007)	*-253.74±0.11259 (0.56745)*
MPSO-1-500	*-83.596±2.0842 (10.504)*	-253.86±0.1265 (0.63751)

B.6 Particle Swarm Optimization with Velocity Adaptation

Detailed results for the experimental investigation of *Particle Swarm Optimization with Velocity Adaptation* (Section 5.3).

Experiment 1

Table B.26: Summary of one-sided Wilcoxon rank sum test with significance level 0.01 for the 100-dimensional benchmarks. For each algorithmic combination (A, B), this matrix shows on which benchmarks A performed significantly better than B. $\mathcal{B}$ denotes the set of all benchmark functions.

100D	1	2	3	4	5
Hyperbolic (1)	{}	{Ro, Ack, Ra, f2, f3, f8, f11, f12, f13, f14}	{Ro, Ack, Ra, Schw, f2, f3, f5, f6, f8, f11, f12, f13, f14}	{f12}	{Schw}
Reflect-S (2)	{Schw, f9, f10}	{}	{Schw, f5, f9, f10, f11}	{Schw, f9, f10}	{Schw, f9, f10}
Infinity-S (3)	{f9, f10}	{Ra, f2, f3, f12, f14}	{}	{}	{Schw}
Reflect-A (4)	{Sph, Ro, Ack, Grie, Ra, Schw, f2, f3, f5, f6, f8, f9, f10, f11, f13}	{Sph, Ro, Ack, Grie, Ra, f2, f3, f5, f6, f8, f11, f12, f13, f14}	$\mathcal{B}\setminus\{f1\}$	{}	{Schw, f5, f9, f10}
Infinity-A (5)	{Sph, Ro, Ack, Grie, Ra, f2, f3, f5, f6, f8, f9, f10, f11, f13}	{Sph, Ro, Ack, Grie, Ra, f2, f3, f6, f8, f11, f12, f13, f14}	$\mathcal{B}\setminus\{Schw, f1\}$	{f2, f3, f12}	{}

Table B.27: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows on which 500-dimensional benchmarks A performed significantly better than B. $\mathcal{B}$ denotes the set of all benchmark functions.

500D	1	2	3	4	5
Hyperbolic (1)	{}	{Sph, Ro, Ack, Grie, Ra, f1, f2, f3, f5, f6, f11, f12, f13, f14}	$\mathcal{B}$	{Schw, f2, f3, f6, f10, f12}	{Schw, f3, f6, f10}
Reflect-S (2)	{Schw, f9, f10}	{}	$\mathcal{B}$	{Schw, f9, f10}	{Schw, f9, f10}
Infinity-S (3)	{}	{}	{}	{}	{}
Reflect-A (4)	{Sph, Ro, Ack, Grie, Ra, f1, f5, f8, f9, f11, f13, f14}	{Sph, Ro, Ack, Grie, Ra, f1, f2, f3, f5, f6, f8, f11, f12, f13, f14}	$\mathcal{B}$	{}	{Schw, f1, f5}
Infinity-A (5)	{Sph, Ro, Ack, Grie, Ra, f1, f2, f5, f8, f9, f11, f13, f14}	{Sph, Ro, Ack, Grie, Ra, f1, f2, f3, f5, f6, f8, f11, f12, f13, f14}	$\mathcal{B}$	{f2, f3, f6, f12}	{}

Table B.28: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated algorithms on various 500-dimensional benchmarks. The best objective values are presented together with the function name.

	Sphere (0)	Rosenbrock (0)
Hyperbolic	7.4707±0.99067 (4.9928)	2791.9±49.433 (249.13)
Reflect-S	1863±250.43 (1262.1)	1141000±306440 (1544400)
Reflect-A	**0.60839±0.23751 (1.197)**	**2218±51.421 (259.15)**
Infinity-S	*765740±16022 (80748)*	*2.3311e+09±43779000 (2.2064e+08)*
Infinity-A	0.75343±0.66991 (3.3762)	2220.6±81.736 (411.93)
	Ackley (0)	Griewank (0)
Hyperbolic	3.3085±0.058416 (0.2944)	0.58759±0.04207 (0.21202)
Reflect-S	10.317±0.25322 (1.2762)	18.033±1.9287 (9.7201)
Reflect-A	1.7404±0.034191 (0.17231)	0.10936±0.030547 (0.15395)
Infinity-S	*20.037±0.022764 (0.11473)*	*7042±62.348 (314.22)*
Infinity-A	**1.7195±0.035576 (0.17929)**	**0.072077±0.023995 (0.12093)**
	Rastrigin (0)	Schwefel (≈-209491)
Hyperbolic	467.81±12.732 (64.167)	-116190±1305.6 (6579.7)
Reflect-S	1979.6±35.599 (179.41)	**-141480±995.72 (5018.2)**
Reflect-A	**420.79±18.884 (95.173)**	-103910±1331.5 (6710.6)
Infinity-S	*6724.2±19.196 (96.745)*	*-11664±418.6 (2109.6)*
Infinity-A	424.15±19.909 (100.34)	-91850±1531.9 (7720.2)
	f1 (-450)	f2 (-450)
Hyperbolic	-417.69±2.2298 (11.238)	954060±12772 (64369)
Reflect-S	2020.4±351.1 (1769.5)	2214800±36523 (184070)
Reflect-A	**-437.65±6.5967 (33.246)**	1037300±25589 (128960)
Infinity-S	*2184400±10131 (51060)*	*90761000±6293900 (3.172e+07)*
Infinity-A	-433.69±4.4911 (22.634)	**8.7e+05±15305 (77133)**

Table B.29: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated algorithms on various 500-dimensional benchmarks. The best objective values are presented together with the function name.

	f3 (-450)	f5 (-310)
Hyperbolic	**4.0784e+08±5140200 (25906000)**	171200±942.14 (4748.2)
Reflect-S	1.5654e+09±43014000 (2.1678e+08)	184960±1821.8 (9181.4)
Reflect-A	4.8178e+08±9056600 (45643000)	**151120±947.57 (4775.6)**
Infinity-S	*1.1425e+11±1.4993e+09 (7.5562e+09)*	*283800±1877.4 (9461.7)*
Infinity-A	4.3839e+08±7318500 (36884000)	153230±846.41 (4265.7)
	f6 (390)	f8 (-140)
Hyperbolic	**49565±4214.7 (21241)**	-118.46±2.4297e-03 (0.012245)
Reflect-S	2.2777e+08±37574000 (1.8936e+08)	-118.46±2.3637e-03 (0.011913)
Reflect-A	589810±218840 (1102900)	**-118.63±3.517e-03 (0.017725)**
Infinity-S	*2.0307e+12±2.4978e+10 (1.2588e+11)*	*-118.42±2.7129e-03 (0.013673)*
Infinity-A	545580±281320 (1417800)	-118.62±3.0479e-03 (0.015361)
	f9 (-330)	f10 (-330)
Hyperbolic	3695.4±28.08 (141.52)	7320±61.405 (309.46)
Reflect-S	**3127.6±58.176 (293.19)**	**4358.3±102.76 (517.89)**
Reflect-A	3348.7±29.11 (146.71)	7806.6±69.867 (352.11)
Infinity-S	*10219±31.573 (159.12)*	*16854±84.02 (423.44)*
Infinity-A	3354.8±28.16 (141.92)	7800.3±50.188 (252.94)
	f11 (-460)	f12 (90)
Hyperbolic	892.79±4.5699 (23.031)	18562000±918840 (4630800)
Reflect-S	909.21±3.0622 (15.433)	62727000±2079600 (10481000)
Reflect-A	761.21±6.155 (31.02)	36247000±2566800 (12936000)
Infinity-S	*1027.8±2.1059 (10.613)*	*1.3232e+09±9181600 (46273000)*
Infinity-A	**757.4±6.4406 (32.459)**	**17962000±610310 (3075800)**
	f13 (-130)	f14 (-300)
Hyperbolic	602.78±15.087 (76.033)	-56.356±0.19687 (0.9922)
Reflect-S	4658.7±605.51 (3051.6)	-54.8±0.083713 (0.42189)
Reflect-A	**417.93±16.504 (83.177)**	**-60.114±0.23181 (1.1683)**
Infinity-S	*46236000±1125500 (5672400)*	*-52.733±0.094504 (0.47628)*
Infinity-A	424.91±18.808 (94.788)	-60.053±0.23051 (1.1617)

Table B.30: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated algorithms on 500-dimensional benchmarks. The best objective values are shown for each function.

	Sphere (0)	Rosenbrock (0)
Nearest-S	1415.7±150.55 (758.75)	497650±87782 (442400)
Random-S	*1483.3±163.7 (825.03)*	*825350±178490 (899570)*
Nearest-A	0.53054±0.2531 (1.2756)	2262.1±66.443 (334.86)
Random-A	**0.44323±0.11685 (0.58891)**	**2206.1±53.97 (272)**
	Ackley (0)	Griewank (0)
Nearest-S	9.477±0.20184 (1.0172)	*15.521±1.6635 (8.3834)*
Random-S	*9.4959±0.20705 (1.0435)*	14.298±1.3265 (6.6851)
Nearest-A	**1.7021±0.037973 (0.19137)**	0.1222±0.03706 (0.18677)
Random-A	1.7279±0.03451 (0.17392)	**0.10632±0.034872 (0.17575)**
	Rastrigin (0)	Schwefel (≈-209491)
Nearest-S	*1929.7±37.692 (189.96)*	**-122280±1228.8 (6192.8)**
Random-S	1649.8±36.477 (183.84)	-108040±970.77 (4892.4)
Nearest-A	422.54±19.263 (97.083)	-100700±1210.1 (6098.5)
Random-A	**404.91±19.154 (96.533)**	*-87882±1333.4 (6720)*
	f1 (-450)	f2 (-450)
Nearest-S	3350.5±578.31 (2914.6)	2425300±51029 (257180)
Random-S	32289±1754.7 (8843.5)	2636100±70381 (354700)
Nearest-A	**-438.78±4.4623 (22.489)**	**1059700±27992 (141070)**
Random-A	*188070±5653.8 (28494)*	*2863700±178850 (901380)*
	f3 (-450)	f5 (-310)
Nearest-S	1.6679e+09±41651000 (2.0991e+08)	*184350±1485.3 (7485.7)*
Random-S	3.5201e+09±1.5178e+08 (7.6494e+08)	180690±1555.2 (7837.7)
Nearest-A	**4.8073e+08±9193900 (46335000)**	**152210±947.5 (4775.2)**
Random-A	*6.5107e+09±1.6452e+08 (8.2914e+08)*	172530±1396.3 (7036.9)
	f6 (390)	f8 (-140)
Nearest-S	6.3049e+08±1.8273e+08 (9.2093e+08)	-118.46±2.4445e-03 (0.01232)
Random-S	1.6815e+09±2.0833e+08 (1.05e+09)	*-118.45±2.0379e-03 (0.010271)*
Nearest-A	**761730±269210 (1356800)**	**-118.63±4.1009e-03 (0.020668)**
Random-A	*4.5844e+09±6.4045e+08 (3.2277e+09)*	**-118.63±3.8841e-03 (0.019575)**
	f9 (-330)	f10 (-330)
Nearest-S	3252.9±52.897 (266.59)	**4754.7±97.199 (489.86)**
Random-S	**3192.1±51.281 (258.45)**	5501.2±102.26 (515.36)
Nearest-A	3326.5±29.679 (149.58)	7766.8±59.283 (298.77)
Random-A	*3435.8±23.449 (118.18)*	*7900.2±53.178 (268.01)*
	f11 (-460)	f12 (90)
Nearest-S	910.95±3.0092 (15.166)	86775000±5069900 (25551000)
Random-S	*920.3±3.0589 (15.416)*	9.226e+07±3528800 (17785000)
Nearest-A	**763.91±4.6449 (23.409)**	**33596000±2429300 (12243000)**
Random-A	841.93±8.1259 (40.953)	*2.0987e+08±9122900 (45977000)*
	f13 (-130)	f14 (-300)
Nearest-S	*10584±2636.3 (13287)*	*-54.774±0.09575 (0.48256)*
Random-S	3749±329.3 (1659.6)	-54.814±0.074409 (0.37501)
Nearest-A	411.11±14.586 (73.508)	**-60.131±0.25707 (1.2956)**
Random-A	**408.61±14.912 (75.154)**	-60.095±0.21916 (1.1045)

Experiment 2

Table B.31: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows on which 100-dimensional benchmarks A performed significantly better than B.

	1	2	3	4	5
Reflect-A-0.01 (1)	{}	{Schw, f5, f8, f9, f10, f11, f14}	{Schw, f5, f8, f9, f10, f11, f13, f14}	{Schw, f5, f8, f9, f10, f11, f13, f14}	{Schw, f5, f8, f9, f10, f11, f13, f14}
Reflect-A-0.1 (2)	{Sph, Ro, Ack, Grie, f1, f2, f3, f6, f12, f13}	{}	{Schw, f5, f8, f9, f13}	{Schw, f5, f6, f8, f9, f10, f13}	{Sph, Schw, f2, f5, f6, f8, f9, f10, f13}
Reflect-A-0.2 (3)	{Sph, Ro, Ack, Grie, Ra, f1, f2, f3, f6, f12}	{Sph, Ack, Grie, f2, f3, f12}	{}	{Sph, Schw, f2, f5, f6, f9, f10, f13}	{Sph, Schw, f2, f3, f5, f6, f9, f10, f12, f13}
Reflect-A-0.5 (4)	{Sph, Ro, Ack, Grie, Ra, f1, f2, f3, f6, f12}	{Ro, Ack, Grie, f3, f12}	{Ack, Grie, f3}	{}	{Ack, Grie, f2, f3, f6, f12}
Reflect-A-0.8 (5)	{Sph, Ro, Ack, Grie, f1, f2, f3, f6, f12}	{Ro, Ack, Grie, f3}	{Ro}	{}	{}

Table B.32: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows how often A performed significantly better than B on the 100-dimensional benchmarks. The total number of benchmark functions is 18.

	1	2	3	4	5
Reflect-A-0.01 (1)	0	7	8	8	8
Reflect-A-0.1 (2)	10	0	5	7	9
Reflect-A-0.2 (3)	10	6	0	8	10
Reflect-A-0.5 (4)	10	5	3	0	6
Reflect-A-0.8 (5)	9	4	1	0	0

Table B.33: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies on various 100-dimensional benchmarks. The best objective values are presented together with the function name.

	Sphere (0)	Rosenbrock (0)
Reflect-A-0.01	*0.22919±0.040874 (0.206)*	*136.89±9.0563 (45.642)*
Reflect-A-0.1	1.1054e-06±1.4453e-08 (7.2842e-08)	**105.59±4.6908 (23.641)**
Reflect-A-0.2	**1.0449e-06±1.3691e-08 (6.9001e-08)**	114.19±7.311 (36.846)
Reflect-A-0.5	1.1551e-06±2.7714e-08 (1.3967e-07)	115.74±7.1559 (36.064)
Reflect-A-0.8	1.1754e-06±3.3247e-08 (1.6756e-07)	113.42±7.1746 (36.159)
	Ackley (0)	Griewank (0)
Reflect-A-0.01	*0.096961±9.6794e-03 (0.048782)*	*0.17687±0.021501 (0.10836)*
Reflect-A-0.1	5.0649e-06±2.7408e-07 (1.3813e-06)	**5.1833e-04±5.6077e-04 (2.8261e-03)**
Reflect-A-0.2	3.6796e-06±2.3598e-08 (1.1893e-07)	6.4195e-04±5.454e-04 (2.7487e-03)
Reflect-A-0.5	**3.131e-06±8.6397e-08 (4.3542e-07)**	1.6515e-03±8.4507e-04 (4.259e-03)
Reflect-A-0.8	3.6282e-06±6.6705e-08 (3.3618e-07)	1.6276e-03±7.8674e-04 (3.965e-03)
	Rastrigin (0)	Schwefel (≈ -41898.3)
Reflect-A-0.01	*101.54±2.9547 (14.891)*	**-31553±246.12 (1240.4)**
Reflect-A-0.1	97.987±3.2882 (16.572)	-26615±272.73 (1374.5)
Reflect-A-0.2	**93.609±3.5117 (17.698)**	-25448±285.17 (1437.2)
Reflect-A-0.5	96.042±3.9068 (19.689)	*-23789±268.31 (1352.2)*
Reflect-A-0.8	99.219±3.8062 (19.182)	-23838±285.24 (1437.6)
	f1 (-450)	f2 (-450)
Reflect-A-0.01	*-449.33±0.12807 (0.64545)*	*22994±575.93 (2902.5)*
Reflect-A-0.1	**-450 (0)**	1685.9±96.253 (485.09)
Reflect-A-0.2	**-450 (0)**	**333.15±33.564 (169.15)**
Reflect-A-0.5	**-450 (0)**	2102±282.69 (1424.7)
Reflect-A-0.8	**-450 (0)**	14435±841.11 (4239)

Table B.34: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated bound handling strategies on various 100-dimensional benchmarks. The best objective values are presented together with the function name.

	f3 (-450)	f5 (-310)
Reflect-A-0.01	*43952000±1802400 (9083900)*	**10118±336.46 (1695.7)**
Reflect-A-0.1	11683000±521300 (2627200)	19322±534.36 (2693)
Reflect-A-0.2	8177500±345810 (1742800)	23575±611.22 (3080.4)
Reflect-A-0.5	**5626700±247740 (1248600)**	*30030±573.97 (2892.7)*
Reflect-A-0.8	8884300±3.6e+05 (1814300)	29710±564.98 (2847.4)
	f6 (390)	f8 (-140)
Reflect-A-0.01	*798.4±43.341 (218.43)*	**-119.17±9.7436e-03 (0.049105)**
Reflect-A-0.1	**500.15±5.1833 (26.123)**	-118.96±8.021e-03 (0.040424)
Reflect-A-0.2	514.39±6.9077 (34.813)	-118.93±8.0908e-03 (0.040776)
Reflect-A-0.5	554.58±9.7745 (49.261)	*-118.92±6.449e-03 (0.032501)*
Reflect-A-0.8	581.78±10.02 (50.498)	-118.93±8.2414e-03 (0.041535)
	f9 (-330)	f10 (-330)
Reflect-A-0.01	**-78.783±9.1213 (45.969)**	**-146.71±7.1081 (35.823)**
Reflect-A-0.1	90.245±9.0306 (45.512)	64.947±10.806 (54.46)
Reflect-A-0.2	119.05±13 (65.515)	59.53±17.302 (87.198)
Reflect-A-0.5	182.24±10.041 (50.602)	377.93±13.664 (68.863)
Reflect-A-0.8	*186.59±8.6135 (43.41)*	*380.57±12.991 (65.471)*
	f11 (-460)	f12 (90)
Reflect-A-0.01	**156.59±1.7268 (8.7028)**	*539250±47092 (237330)*
Reflect-A-0.1	184.73±1.724 (8.6887)	337680±62921 (317110)
Reflect-A-0.2	184.69±1.5947 (8.0371)	**238170±34788 (175330)**
Reflect-A-0.5	184.59±1.5609 (7.8665)	264740±44208 (222800)
Reflect-A-0.8	*185.14±1.5175 (7.6479)*	453360±92842 (467900)
	f13 (-130)	f14 (-300)
Reflect-A-0.01	-114.44±0.45918 (2.3142)	**-254.64±0.1177 (0.59317)**
Reflect-A-0.1	**-115.3±0.50791 (2.5598)**	-254.05±0.10543 (0.53133)
Reflect-A-0.2	-112.08±0.63547 (3.2026)	-254.1±0.11193 (0.56408)
Reflect-A-0.5	*-105.92±0.96466 (4.8617)*	-254.05±0.10784 (0.54351)
Reflect-A-0.8	-106.26±0.98447 (4.9615)	*-253.95±0.11036 (0.5562)*

Experiment 3

Table B.35: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows on which 100-dimensional benchmarks A performed significantly better than B.

	1	2	3	4
Reflect-A (1)	{}	{Schw, f5, f9, f10}	{Sph, Ro, Ack, Schw, f2, f5, f6, f8, f9, f10, f11, f13, f14}	{Sph, Ro, Ack, Grie, Schw, f2, f5, f6, f8, f9, f10, f11, f13, f14}
Infinity-A (2)	{f2, f3, f12}	{}	{Sph, Ro, Ack, f2, f3, f6, f8, f9, f10, f11, f12, f13, f14}	{Sph, Ro, Ack, Grie, f2, f5, f6, f8, f9, f10, f11, f13, f14}
Reflect-A-maxvel (3)	{}	{Schw}	{}	{Schw, f5, f8}
Infinity-A-maxvel (4)	{f3, f12}	{}	{f12}	{}

Table B.36: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated algorithms on 500-dimensional benchmarks. The best objective values are shown for each function.

	Sphere (0)	Rosenbrock (0)
Reflect-A	**1.0449e-06±1.3691e-08 (6.9001e-08)**	**114.19±7.311 (36.846)**
Reflect-A-maxvel	1.0804e-06±1.2126e-08 (6.1111e-08)	155.53±9.2666 (46.702)
Infinity-A	1.0519e-06±1.3771e-08 (6.9402e-08)	123.29±7.7796 (39.207)
Infinity-A-maxvel	*1.0911e-06±1.1819e-08 (5.9563e-08)*	*163±9.2017 (46.375)*
	Ackley (0)	Rastrigin (0)
Reflect-A	**3.6796e-06±2.3598e-08 (1.1893e-07)**	93.609±3.5117 (17.698)
Reflect-A-maxvel	1.0716±0.17175 (0.8656)	*96.637±3.8506 (19.406)*
Infinity-A	3.7034e-06±2.469e-08 (1.2443e-07)	**91.612±2.9265 (14.749)**
Infinity-A-maxvel	*1.2136±0.17233 (0.8685)*	94.721±3.4147 (17.209)
	Griewank (0)	Schwefel (≈ -41898.3)
Reflect-A	**6.4195e-04±5.454e-04 (2.7487e-03)**	**-25448±285.17 (1437.2)**
Reflect-A-maxvel	2.4367e-03±1.2749e-03 (6.4253e-03)	-24274±359.74 (1813)
Infinity-A	1.0854e-03±6.7476e-04 (3.4007e-03)	*-22832±222.92 (1123.5)*
Infinity-A-maxvel	*6.2926e-03±4.4086e-03 (0.022218)*	-22911±336.02 (1693.5)
	f1 (-450)	f2 (-450)
Reflect-A	**-450 (0)**	333.15±33.564 (169.15)
Reflect-A-maxvel	**-450 (0)**	*771.77±133.27 (671.67)*
Infinity-A	**-450 (0)**	**264.55±36.152 (182.2)**
Infinity-A-maxvel	**-450 (0)**	643.01±73.067 (368.24)
	f3 (-450)	f5 (-310)
Reflect-A	*8177500±345810 (1742800)*	**23575±611.22 (3080.4)**
Reflect-A-maxvel	7718900±323750 (1631600)	27488±623.24 (3141)
Infinity-A	**7138300±279290 (1407600)**	26909±518.5 (2613.1)
Infinity-A-maxvel	7385100±301490 (1519500)	*29447±557.35 (2808.9)*
	f6 (390)	f8 (-140)
Reflect-A	**514.39±6.9077 (34.813)**	**-118.93±8.0908e-03 (0.040776)**
Reflect-A-maxvel	*564.08±9.4243 (47.496)*	-118.69±6.7715e-03 (0.034127)
Infinity-A	515.31±7.3591 (37.088)	-118.92±8.0686e-03 (0.040664)
Infinity-A-maxvel	559.68±9.3655 (47.2)	*-118.68±5.82e-03 (0.029331)*
	f9 (-330)	f10 (-330)
Reflect-A	**119.05±13 (65.515)**	**59.53±17.302 (87.198)**
Reflect-A-maxvel	237.04±10.884 (54.855)	491±20.417 (102.9)
Infinity-A	159.68±12.282 (61.898)	174.39±11.563 (58.275)
Infinity-A-maxvel	*239.08±9.8974 (49.881)*	*518.75±20.65 (104.07)*
	f11 (-460)	f12 (90)
Reflect-A	**184.69±1.5947 (8.0371)**	238170±34788 (175330)
Reflect-A-maxvel	213.4±1.8366 (9.2559)	*270440±36112 (182000)*
Infinity-A	184.9±1.7212 (8.6746)	**163930±16332 (82310)**
Infinity-A-maxvel	*213.61±1.5986 (8.0565)*	170170±15230 (76754)
	f13 (-130)	f14 (-300)
Reflect-A	-112.08±0.63547 (3.2026)	**-254.1±0.11193 (0.56408)**
Reflect-A-maxvel	-98.013±1.6086 (8.1069)	*-253.15±0.088931 (0.44819)*
Infinity-A	**-112.41±0.58494 (2.948)**	-254.02±0.10392 (0.52373)
Infinity-A-maxvel	*-97.431±1.489 (7.504)*	-253.21±0.089066 (0.44887)

Experiment 4

Table B.37: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows on which 100-dimensional benchmarks A performed significantly better than B.

	1	2	3	4
Reflect-A (1)	{}	{Schw, f5, f9, f10}	{Ro, f6, f9, f10}	{Ro, Schw, f5, f9, f10}
Infinity-A (2)	{f2, f3, f12}	{}	{f2, f3, f6, f12}	{f9, f10}
Reflect-A-ind (3)	{Ack, Grie, f3, f8}	{Ack, Grie, Schw, f5, f8, f14}	{}	{Schw, f5, f9, f10}
Infinity-A-ind (4)	{Ack, Grie, f3, f8, f12}	{Ack, Grie, f8}	{f2, f12}	{}

Table B.38: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated algorithms on 500-dimensional benchmarks. The best objective values are shown for each function.

	Sphere (0)	Rosenbrock (0)
Reflect-A	1.0449e-06±1.3691e-08 (6.9001e-08)	**114.19±7.311 (36.846)**
Reflect-A-ind	**1.0426e-06±1.3855e-08 (6.9826e-08)**	132.15±7.7635 (39.126)
Infinity-A	*1.0519e-06±1.3771e-08 (6.9402e-08)*	123.29±7.7796 (39.207)
Infinity-A-ind	1.0454e-06±1.4205e-08 (7.1591e-08)	*132.81±8.4945 (42.81)*
	Ackley (0)	Griewank (0)
Reflect-A	**3.6796e-06±2.3598e-08 (1.1893e-07)**	**6.4195e-04±5.454e-04 (2.7487e-03)**
Reflect-A-ind	0.027945±0.03906 (0.19685)	1.0112e-03±6.4464e-04 (3.2488e-03)
Infinity-A	3.7034e-06±2.469e-08 (1.2443e-07)	1.0854e-03±6.7476e-04 (3.4007e-03)
Infinity-A-ind	*0.05677±0.049677 (0.25036)*	*1.2083e-03±6.822e-04 (3.4381e-03)*
	Rastrigin (0)	Schwefel (≈ -41898.3)
Reflect-A	*93.609±3.5117 (17.698)*	**-25448±285.17 (1437.2)**
Reflect-A-ind	92.161±3.2946 (16.604)	-25371±293.13 (1477.3)
Infinity-A	91.612±2.9265 (14.749)	*-22832±222.92 (1123.5)*
Infinity-A-ind	**91.576±2.9515 (14.875)**	-23059±282.92 (1425.8)
	f1 (-450)	f2 (-450)
Reflect-A	**-450 (0)**	333.15±33.564 (169.15)
Reflect-A-ind	**-450 (0)**	*364.79±36.791 (185.42)*
Infinity-A	**-450 (0)**	**264.55±36.152 (182.2)**
Infinity-A-ind	**-450 (0)**	279.23±28.788 (145.09)
	f3 (-450)	f5 (-310)
Reflect-A	*8177500±345810 (1742800)*	**23575±611.22 (3080.4)**
Reflect-A-ind	7674300±275900 (1390500)	23812±574.37 (2894.7)
Infinity-A	**7138300±279290 (1407600)**	26909±518.5 (2613.1)
Infinity-A-ind	7287000±244520 (1232300)	*27550±482.58 (2432.1)*
	f6 (390)	f8 (-140)
Reflect-A	**514.39±6.9077 (34.813)**	-118.93±8.0908e-03 (0.040776)
Reflect-A-ind	*532.42±8.6171 (43.428)*	**-119.01±8.9517e-03 (0.045115)**
Infinity-A	515.31±7.3591 (37.088)	*-118.92±8.0686e-03 (0.040664)*
Infinity-A-ind	519.7±7.3949 (37.269)	**-119.01±8.1445e-03 (0.041046)**
	f9 (-330)	f10 (-330)
Reflect-A	**119.05±13 (65.515)**	**59.53±17.302 (87.198)**
Reflect-A-ind	166.14±9.5308 (48.033)	176.89±19.787 (99.72)
Infinity-A	159.68±12.282 (61.898)	174.39±11.563 (58.275)
Infinity-A-ind	*196.07±9.5977 (48.37)*	*304.85±19.506 (98.307)*
	f11 (-460)	f12 (90)
Reflect-A	**184.69±1.5947 (8.0371)**	238170±34788 (175330)
Reflect-A-ind	*186.11±1.5179 (7.6499)*	*265960±34752 (175140)*
Infinity-A	184.9±1.7212 (8.6746)	163930±16332 (82310)
Infinity-A-ind	185.53±1.5777 (7.9512)	**158010±16108 (81182)**
	f13 (-130)	f14 (-300)
Reflect-A	*-112.08±0.63547 (3.2026)*	-254.1±0.11193 (0.56408)
Reflect-A-ind	-112.26±0.73385 (3.6984)	**-254.23±0.10825 (0.54553)**
Infinity-A	**-112.41±0.58494 (2.948)**	*-254.02±0.10392 (0.52373)*
Infinity-A-ind	-112.19±0.58343 (2.9404)	-254.19±0.1239 (0.62441)

Experiment 5

Table B.39: Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows on which 100-dimensional benchmarks A performed significantly better than B.

	1	2	3	4
Reflect-A (1)	{}	{Schw, f5, f9, f10}	{Schw, f2, f3, f5, f9, f10, f11, f12, f13, f14}	{Schw, f5, f9, f10, f11, f12, f13, f14}
Infinity-A (2)	{f2, f3, f12}	{}	{Schw, f2, f3, f5, f9, f10, f11, f12, f13, f14}	{Schw, f2, f3, f5, f9, f10, f11, f12, f13, f14}
Reflect-A-init2 (3)	{Sph, Ack, Grie}	{Sph, Ack, Grie}	{}	{f5}
Infinity-A-init2 (4)	{Sph, Ack, Grie}	{Sph, Ack, Grie}	{f10, f12}	{}

Table B.40: Sample mean of final objective values, 95% confidence intervals, and standard deviations (in brackets) of the investigated algorithms on 500-dimensional benchmarks. The best objective values are shown for each function.

	Sphere (0)	Rosenbrock (0)
Reflect-A	1.0449e-06±1.3691e-08 (6.9001e-08)	**114.19±7.311 (36.846)**
Reflect-A-init2	**8.9654e-07±9.4342e-09 (4.7546e-08)**	123.27±7.5751 (38.177)
Infinity-A	*1.0519e-06±1.3771e-08 (6.9402e-08)*	*123.29±7.7796 (39.207)*
Infinity-A-init2	8.978e-07±1.0063e-08 (5.0717e-08)	121.83±6.9907 (35.231)
	Ackley (0)	Griewank (0)
Reflect-A	3.6796e-06±2.3598e-08 (1.1893e-07)	**6.4195e-04±5.454e-04 (2.7487e-03)**
Reflect-A-init2	3.2161e-06±2.1491e-08 (1.0831e-07)	1.2581e-03±6.3892e-04 (3.22e-03)
Infinity-A	*3.7034e-06±2.469e-08 (1.2443e-07)*	1.0854e-03±6.7476e-04 (3.4007e-03)
Infinity-A-init2	**3.2086e-06±2.2626e-08 (1.1403e-07)**	*1.3737e-03±1.4011e-03 (7.0611e-03)*
	Rastrigin (0)	Schwefel (≈ -41898.3)
Reflect-A	93.609±3.5117 (17.698)	**-25448±285.17 (1437.2)**
Reflect-A-init2	*95.964±4.9064 (24.727)*	-15988±659.48 (3323.6)
Infinity-A	**91.612±2.9265 (14.749)**	-22832±222.92 (1123.5)
Infinity-A-init2	92.595±4.8184 (24.283)	*-15248±512.28 (2581.8)*
	f1 (-450)	f2 (-450)
Reflect-A	**-450 (0)**	333.15±33.564 (169.15)
Reflect-A-init2	**-450 (0)**	410.29±40.49 (204.06)
Infinity-A	**-450 (0)**	**264.55±36.152 (182.2)**
Infinity-A-init2	**-450 (0)**	*411.56±48.885 (246.37)*
	f3 (-450)	f5 (-310)
Reflect-A	8177500±345810 (1742800)	**23575±611.22 (3080.4)**
Reflect-A-init2	*9928000±633720 (3193800)*	30655±774.25 (3902)
Infinity-A	**7138300±279290 (1407600)**	26909±518.5 (2613.1)
Infinity-A-init2	8746400±458740 (2311900)	*32721±677.48 (3414.3)*
	f6 (390)	f8 (-140)
Reflect-A	514.39±6.9077 (34.813)	-118.93±8.0908e-03 (0.040776)
Reflect-A-init2	*515.67±7.8349 (39.486)*	-118.93±8.8913e-03 (0.04481)
Infinity-A	515.31±7.3591 (37.088)	*-118.92±8.0686e-03 (0.040664)*
Infinity-A-init2	**514.13±7.1859 (36.215)**	**-118.94±8.54e-03 (0.04304)**
	f9 (-330)	f10 (-330)
Reflect-A	**119.05±13 (65.515)**	**59.53±17.302 (87.198)**
Reflect-A-init2	415.33±8.1107 (40.876)	*1025.8±14.935 (75.271)*
Infinity-A	159.68±12.282 (61.898)	174.39±11.563 (58.275)
Infinity-A-init2	*419.58±8.5384 (43.031)*	996.52±14.7 (74.086)
	f11 (-460)	f12 (90)
Reflect-A	**184.69±1.5947 (8.0371)**	238170±34788 (175330)
Reflect-A-init2	247.62±0.89955 (4.5335)	*389730±50415 (254080)*
Infinity-A	184.9±1.7212 (8.6746)	**163930±16332 (82310)**
Infinity-A-init2	*248.18±0.90982 (4.5853)*	283700±31071 (156590)
	f13 (-130)	f14 (-300)
Reflect-A	-112.08±0.63547 (3.2026)	**-254.1±0.11193 (0.56408)**
Reflect-A-init2	*-104.04±1.1319 (5.7044)*	-253.04±0.0848 (0.42737)
Infinity-A	**-112.41±0.58494 (2.948)**	-254.02±0.10392 (0.52373)
Infinity-A-init2	-105.13±1.1648 (5.8704)	*-253.01±0.073852 (0.3722)*

Bibliography

[ABEF05] Julio E. Alvarez-Benitez, Richard M. Everson, and Jonathan E. Fieldsend. A MOPSO algorithm based exclusively on pareto dominance concepts. In *Evolutionlary Multi-Criterion Optimization*, pages 459–473. Springer, 2005.

[Abi02] Mohammad A. Abido. Optimal power flow using particle swarm optimization. *International Journal of Electrical Power & Energy Systems*, 24(7):563–571, 2002.

[ACXZ05] Jiyuan An, Yi-Ping P. Chen, Qinying Xu, and Xiaofang Zhou. A new indexing method for high dimensional dataset. In *DASFAA*, volume 3453, pages 385–397. Springer, 2005.

[AHK01] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Database Theory – ICDT 2001*, pages 420–434. Springer, 2001.

[AP09] Davide Anghinolfi and Massimo Paolucci. A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 193(1):73–85, 2009.

[AT07] Anne Auger and Olivier Teytaud. Continuous lunches are free! In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 916–922. ACM, 2007.

[BDT99] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: From natural to artificial systems.* Oxford University Press, 1999.

[Bel61] Richard Bellman. *Adaptive control processes: A guided tour.* Princeton University Press, 1961.

[Ber95] Dimitri P. Bertsekas. *Nonlinear optimization.* Athena Scientific, 1995.

[Bey00] Hans-Georg Beyer. Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):239–267, 2000.

[BF05] Bogdan Bochenek and Pawel Foryś. Structural optimization against instability using particle swarms. In *6th World Congress on Structural and Multidisciplinary Optimization*, 2005. 10 pages, CD-ROM Proceedings.

[BG02] David Bradley and Ramesh Gupta. On the distribution of the sum of *n* non-identically distributed uniform random variables. *Annals of the Institute of Statistical Mathematics*, 54(3):689–700, 2002.

[BGK$^+$95] Richard S. Barr, Bruce L. Golden, James P. Kelly, Mauricio G.C. Resende, and William R. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32, 1995.

[BK07] Daniel Bratton and James Kennedy. Defining a standard for particle swarm optimization. In *Proceedings of the 2007 IEEE Swarm Intelligence Symposium*, pages 120–127, 2007.

[BM92] Paul J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[BM06] Jürgen Branke and Sanaz Mostaghim. About selecting the personal best in multi-objective particle swarm optimization. In *Proceedings of Parallel Problem Solving from Nature – PPSN IX*, pages 523–532. Springer, 2006.

[BS02] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[BSMD08] Sanghamitra Bandyopadhyay, Sriparna Saha, Ujjwal Maulik, and Kalyanmoy Deb. A simulated annealing-based multiobjective optimization algorithm: AMOSA. *IEEE Transactions on Evolutionary Computation*, 12(3):269–283, 2008.

[BT78] Herbert Büning and Götz Trenkler. *Nichtparametrische statistische Methoden*. de Gruyter, 1978.

[BTT98] Tobias Blickle, Jürgen Teich, and Lothar Thiele. System-level synthesis using evolutionary algorithms. *J. Design Automation for Embedded Systems*, 3(1):23–58, 1998.

[C$^+$07] Maurice Clerc et al. Standard PSO 2007 (standard_pso_2007.c). http://www.particleswarm.info, 2007.

[CCZ09] Zhihua Cui, Xingjuan Cai, and Jianchao Zeng. Stochastic velocity threshold inspired by evolutionary programming. In *Proceedings of the 2009 World Congress on Nature & Biologically Inspired Computing*, pages 626 – 631, 2009.

[CDG99] David Corne, Marco Dorigo, and Fred Glover, editors. *New ideas in optimization*. McGraw-Hill, London, UK, 1999.

[CE08] Douglas Curran-Everett. Explorations in statistics: Standard deviations and standard errors. *Advances in Physiology Education*, 32:203–208, 2008.

[CE09] Douglas Curran-Everett. Explorations in statistics: Hypothesis tests and p values. *Advances in Physiology Education*, 33:81–86, 2009.

[CETK98] Douglas Curran-Everett, Sue Taylor, and Karen Kafadar. Fundamental concepts in statistics: Elucidation and illustration. *Journal of Applied Physiology*, 85:775–786, 1998.

[CK02] Maurice Clerc and James Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[CK03] David Corne and Joshua Knowles. Some multiobjective optimizers are better than others. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, volume 4, pages 2506–2512, 2003.

[Cle00] Maurice Clerc. Discrete particle swarm optimization – illustrated by the traveling salesman problem. http://clerc.maurice.free.fr/pso/, 2000.

[Cle03] Maurice Clerc. Tribes, a parameter free particle swarm optimizer. http://clerc.maurice.free.fr/pso/ (last checked: 22.12.2008), 2003.

[Cle06a] Maurice Clerc. Confinements and biases in particle swarm optimization. http://clerc.maurice.free.fr/pso/, 2006.

[Cle06b] Maurice Clerc. *Particle swarm optimization*. ISTE, 2006.

[Cle06c] Maurice Clerc. Stagnation analysis in particle swarm optimization or what happens when nothing happens. Technical Report CSM–460, Department of Computer Science, University of Essex, 2006.

[Cle07] Maurice Clerc. Back to random topology. http://clerc.maurice.free.fr, 2007.

[Coe02a] Carlos A. Coello Coello. MOPSO: A proposal for multiple objective particle swarm optimization. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, pages 1051–1056, 2002.

[Coe02b] Carlos A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12):1245–1287, 2002.

[CPL04] Carlos A. Coello Coello, Gregorio Toscano Pulido, and Maximino Salazar Lechuga. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279, 2004.

[CVL02] Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary algorithms for solving multi-objective problems*. Kluwer Academic Publishers, 2002.

[CZS06] Zhihua Cui, Jianchao Zeng, and Guoji Sun. Adaptive velocity threshold particle swarm optimization. In *Rough Sets and Knowledge Technology*. Springer, 2006.

[DAPM02] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[Deb00] Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):311–338, 2000.

[Deb01] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley and Sons, 2001.

[DG97] Marco Dorigo and Luca Maria Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions Evolutionary Computation*, 1(1):53–66, 1997.

[DMC91] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Positive feedback as a search strategy. Technical Report 91–016, Politecnico di Milano, Italy, 1991.

[DS04] Marco Dorigo and Thomas Stützle. *Ant colony optimization*. MIT Press, Cambridge, MA, 2004.

[dS08] Leandro dos Santos Coelho. A quantum particle swarm optimizer with chaotic mutation operator. *Chaos, Solitons & Fractals*, 37(5):1409–1418, 2008.

[DTLZ02] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable multi-objective optimization test problems. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, pages 825–830, 2002.

[Ehr05] Matthias Ehrgott. *Multicriteria optimization*. Springer, 2005.

[EK95] Russell C. Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[EKS04] Matthias Ehrgott, Kathrin Klamroth, and Christian Schwehm. An MCDM approach to portfolio optimization. *European Journal of Operational Research*, 155(3):752–770, 2004.

[EM97] Thomas Eiter and Heikki Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34(2):109–133, 1997.

[Eng05] Andries P. Engelbrecht. *Fundamentals of computational swarm intelligence*. John Wiley and Sons, 2005.

[ES00] Russell C. Eberhart and Yuhui Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 84–88, 2000.

[Fan02] Huiyuan Fan. A modification to particle swarm optimization algorithm. *Engineering Computations*, 19(8):970–989, 2002.

[FES03] Jonathan Fieldsend, Richard M. Everson, and Sameer Singh. Using unconstrained elite archives for multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 7(3):305–323, 2003.

[FG02] P. C. Fourie and A. A. Groenwold. The particle swarm optimization algorithm in size and shape optimization. *Structural and Multidisciplinary Optimization*, 23(4):259–267, 2002.

[Fog62] L.J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[For01] Otto Forster. *Analysis 1 – Differential- und Integralrechnung einer Veränderlichen*. Vieweg, 6th edition, 2001.

[FOW66] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial intelligence through simulated evolution*. John Wiley and Sons, Inc., New York, 1966.

[FS02] Jonathan E. Fieldsend and Sameer Singh. A multi-objective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence. In *Proceedings of the 2002 U.K. Workshop on Computational Intelligence*, pages 37–44, 2002.

[FWV07] Damien François, Vincent Wertz, and Michel Verleysen. The concentration of fractional distances. *IEEE Transactions on Knowledge and Data Engineering*, 19(7):873–886, 2007.

[GAJP89] S. Goss, S. Aron, J.L.Deneubourg, and J.M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

[GMCH07] Carlos García-Martínez, Oscar Cordón, and Francisco Herrera. A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. *European Journal of Operational Research*, 180(1):116–148, 2007.

[Gol89] David E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley Publishing Company, 1989.

[GWHK09] Alexandr Gnezdilov, Stefan Wittmann, Sabine Helwig, and Gabriella Kókai. Acceleration of a relative positioning framework. *International Journal of Computational Intelligence Research (IJCIR)*, 5(2):130–140, 2009.

[HC09] Chih-Ming Hsu and Ming-Syan Chen. On the design and applicability of distance functions in high-dimensional data space. *IEEE Transactions on Knowledge and Data Engineering*, 21(4):523–536, 2009.

[HE02a] Xiaohui Hu and Russell Eberhart. Multiobjective optimization using dynamic neighborhood particle swarm optimization. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, pages 1677–1681, 2002.

[HE02b] Xiaohui Hu and Russell Eberhart. Solving constrained nonlinear optimization problems with particle swarm optimization. In *6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)*, pages 203–206, 2002.

[HES03a] Xiaohui Hu, Russell C. Eberhart, and Yuhui Shi. Engineering optimization with particle swarm. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pages 53–57, 2003.

[HES03b] Xiaohui Hu, Russell C. Eberhart, and Yuhui Shi. Swarm intelligence for permutation optimization: A case study of n-queens problem. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pages 243–246, 2003.

[HG90] Frank Heppner and Ulf Grenander. A stochastic nonlinear model for coordinated bird flocks. In Saul Krasner, editor, *The Ubiquity of Chaos*. American Association for the Advancement of Science, 1990.

[HHBW06] Simon Huband, Phil Hingston, Luigi Barone, and Lyndon While. A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, 10(5):477–506, 2006.

[HKR93] Daniel P. Huttenlocher, Gregory A. Klanderman, and William J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[HM06] Werner Halter and Sanaz Mostaghim. Bilevel optimization of multi-component chemical systems using particle swarm optimization. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 1240–1247, 2006.

[HNW09] Sabine Helwig, Frank Neumann, and Rolf Wanka. Particle swarm optimization with velocity adaptation. In *Proceedings of the International Conference on Adaptive and Intelligent Systems (ICAIS09)*, pages 146–151. IEEE, 2009.

[HNW10] Sabine Helwig, Frank Neumann, and Rolf Wanka. Velocity adaptation in particle swarm optimization. In *Handbook of Swarm Intelligence – Concepts, Principles and Applications*. Springer, 2010. To appear.

[HO96] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of the 1996 International Conference on Evolutionary Computation*, pages 312–317, 1996.

[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[Hol62] John H. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9(3):297–314, 1962.

[Hol75] John H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, 1975.

[Hol79] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

[HS06] Jürgen Hedderich and Lothar Sachs. *Angewandte Statistik – Methodensammlung mit R*. Springer, 2006.

[HW07] Sabine Helwig and Rolf Wanka. Particle swarm optimization in high-dimensional bounded search spaces. In *Proceedings of the 2007 IEEE Swarm Intelligence Symposium*, pages 198–205, 2007.

[HW08] Sabine Helwig and Rolf Wanka. Theoretical analysis of initial particle swarm behavior. In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN08)*, pages 889–898. Springer, 2008.

[Jäg02] Jens Jägersküpper. Analysis of a simple evolutionary algorithm for the minimization in Euclidian spaces. Technical Report CI 140/02, SFB 531, Universität Dortmund, 2002.

[JHW08] Johannes Jordan, Sabine Helwig, and Rolf Wanka. Social interaction in particle swarm optimization, the ranked FIPS, and adaptive multi-swarms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO08)*, pages 49–56. ACM Press, 2008.

[Jin05] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.

[JLY07a] M. Jiang, Y. P. Luo, and S. Y. Yang. Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. *Information Processing Letters*, 102(1):8–16, 2007.

[JLY07b] Ming Jiang, Yupin Luo, and Shiyuan Yang. Stagnation analysis in particle swarm optimization. In *Proceedings of the 2007 IEEE Swarm Intelligence Symposium*, pages 92–99, 2007.

[JM05] Stefan Janson and Martin Middendorf. A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(6):1272–1282, 2005.

[KC03] Joshua Knowles and David Corne. Properties of an adaptive archiving algorithm for storing nondominated vectors. *IEEE Transactions on Evolutionary Computation*, 7(2):100–116, 2003.

[KE95] James Kennedy and Russell C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[KE97] James Kennedy and Russell C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, volume 5, pages 4104–4108, 1997.

[KE01] James Kennedy and Russell C. Eberhart. *Swarm intelligence*. Morgan Kaufmann, 2001.

[Ken99] James Kennedy. Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 3, pages 1931–1938. IEEE Press, 1999.

[Ken00] James Kennedy. Stereotyping: Improving particle swarm performance with cluster analysis. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 2, pages 1507–1512. IEEE Press, 2000.

[Ken03] James Kennedy. Bare bones particle swarms. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pages 80–87, 2003.

[Ken07] James Kennedy. Some issues and practices for particle swarms. In *Proceedings of the 2007 IEEE Swarm Intelligence Symposium*, pages 162–169, April 2007.

[KGV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecci. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KM98] Slawomir Koziel and Zbigniew Michalewicz. A decoder-based evolutionary algorithm for constrained parameter optimization problems. In *Proceedings of Parallel Problem Solving from Nature – PPSN V*, pages 231–240. Springer, 1998.

[KM99] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[KM02] James Kennedy and Rui Mendes. Population structure and particle swarm performance. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1671–1676, 2002.

[KM06] James Kennedy and Rui Mendes. Neighborhood topologies in fully informed and best-of-neighborhood particle swarms. *IEEE Transactions Systems, Man and Cybernetics*, 36(4):515–519, 2006.

[KMN$^+$02] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[Knu97] Donald E. Knuth. *The art of computer programming, volume 1*. Addison Wesley, 3rd edition, 1997.

[Koz92] J.R. Koza. *Genetic programming: On the programming of computers by means of natural selection*. MIT Press, 1992.

[KSF06] Visakan Kadirkamanathan, Kirusnapillai Selvarajah, and Peter J. Fleming. Stability analysis of the particle dynamics in particle swarm optimizer. *IEEE Transactions on Evolutionary Computation*, 10(3):245–255, 2006.

[Lap90] Lawrence L. Lapin. *Probability and statistics for modern engineering*. PWS-KENT Publishing Company, 2nd edition, 1990.

[Lau03] Marco Laumanns. *Analysis and applications of evolutionary multiobjective optimization algorithms*. PhD thesis, ETH Zürich, Swiss, August 2003.

[Llo82] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[LS05] Jing J. Liang and Ponnuthurai N. Suganthan. Dynamic multi-swarm particle swarm optimizer. In *Proceedings of the 2005 Congress on Evolutionary Computation*, volume 1, pages 522–528. IEEE Press, 2005.

[LS06] Jing J. Liang and Ponnuthurai N. Suganthan. Dynamic multi-swarm particle swarm optimizer with a novel constraint-handling mechanism. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 9–16, 2006.

[LTDZ02] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining convergence and diversity in evolutionary multi-objective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.

[MC99] Jacqueline Moore and Richard Chapman. Application of particle swarm to multiobjective optimization. Department of Computer Science and Software Engineering, Auburn University, 1999.

[Men04] Rui Mendes. *Population topologies and their influence in particle swarm performance*. PhD thesis, Departamento de Informática, Escola de Engenharia, Universidade do Minho, 2004.

[MF02a] Vladimiro Miranda and Nuno Fonseca. EPSO – best-of-two-worlds meta-heuristic applied to power system problems. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, volume 2, pages 1080–1085, 2002.

[MF02b] Vladimiro Miranda and Nuno Fonseca. New evolutionary particle swarm algorithm (EPSO) applied to voltage/var control. In *Proceedings of the 14th Power Systems Computation Conference*, 2002.

[Mic00a] Zbigniew Michalewicz. Decoders. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Evolutionary Computation 2 – Advanced Algorithms and Operators*, chapter 8. Institute of Physics Publishing, 2000.

[Mic00b] Zbigniew Michalewicz. Other constraint-handling methods. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Evolutionary Computation 2 – Advanced Algorithms and Operators*, chapter 11. Institute of Physics Publishing, 2000.

[Mic00c] Zbigniew Michalewicz. Repair algorithms. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Evolutionary Computation 2 – Advanced Algorithms and Operators*, chapter 9. Institute of Physics Publishing, 2000.

[MK05] Said Mikki and Ahmed Kishk. Improved particle swarm optimization technique using hard boundary conditions. *Microwave and Optical Technology Letters*, 46(5):422–426, 2005.

[MKN03] Rui Mendes, James Kennedy, and José Neves. Watch thy neighbor or how the swarm can learn from its environment. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 88–94, 2003.

[MKN04] Rui Mendes, James Kennedy, and José Neves. The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, 2004.

[MMWP05] Arvind S. Mohais, Rui Mendes, Christopher Ward, and Christian Posthoff. Neighborhood re-structuring in particle swarm optimization. In *Proceedings of the 18th Australian Joint Conference on Artificial Intelligence*, pages 776–785. Springer, 2005.

[MN95] Zbigniew Michalewicz and Girish Nazhiyath. Genocop III: A co-evolutionary algorithm for numerical optimization problems with non-linear constraints. In *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*, volume 2, pages 647–651, 1995.

[MN04] Rui Mendes and José Neves. What makes a successful society? Experiments with population topologies in particle swarms. In *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence (SBIA 2004)*, pages 346–355. Springer, 2004.

[Mor03] Matthew D. Moran. Arguments for rejecting the sequential Bonferroni in ecological studies. *OIKOS*, 100(2):403–405, 2003.

[MS96] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[MS05] Christopher K. Monson and Kevin D. Seppi. Linear equality constraints and homomorphous mappings in PSO. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 73–80, 2005.

[MT03] Sanaz Mostaghim and Jürgen Teich. Strategies for finding good local guides in multi-objective particle swarm optimization. In *Swarm Intelligence Symposium*, 2003.

[MW47] H.B. Mann and D.R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[MWP04] Arvind S. Mohais, Christopher Ward, and Christian Posthoff. Randomized directed neighborhoods with edge migration in particle swarm optimization. In *Proceedings of the 2004 Congress on Evolutionary Computation*, volume 1, pages 548–555. IEEE Press, 2004.

[NT95] Swami D. Nigam and Joshua U. Turner. Review of statistical approaches to tolerance analysis. *Computer-Aided Design*, 27(1):6–15, 1995.

[OGH94] Andreas Ostermeier, Andreas Gawelczyk, and Nikolaus Hansen. Step-size adaptation based on non-local use of selection information. In *Parallel Problem Solving from Nature – PPSN III*, pages 189–198. Springer, 1994.

[OH07] Alan Owen and Inman Harvey. Adapting particle swarm optimisation for fitness landscapes with neutrality. In *Proceedings of the 2007 IEEE Swarm Intelligence Symposium*, pages 258–265, 2007.

[OJOS04] Tatsuya Okabe, Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. On test functions for evolutionary multi-objective optimization. In *Proceedings of Parallel Problem Solving from Nature – PPSN VIII*, pages 792–802. Springer, 2004.

[OM88] Bernard Ostle and Linda C. Malone. *Statistics in research – basic concepts and techniques for research workers*. Iowa State University Press, 4th edition, 1988.

[OM99] Ender Ozcan and Chilukuri K. Mohan. Particle swarm optimization: Surfing the waves. In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, volume 3, pages 1939–1944, 1999.

[PB07] Riccardo Poli and David Broomhead. Exact analysis of the sampling distribution for the canonical particle swarm optimiser and its convergence during stagnation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO07)*, pages 134–141. ACM Press, 2007.

[PBBK07] Riccardo Poli, Dan Bratton, Tim Blackwell, and James Kennedy. Theoretical derivation, analysis and empirical evaluation of a simpler particle swarm optimizer. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, pages 1955–1962, 2007.

[PC04] Gregorio Toscano Pulido and Carlos A. Coello Coello. A constraint-handling mechanism for particle swarm optimization. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, volume 2, pages 1396–1403, 2004.

[PCSQ07] Gregorio Toscano Pulido, Carlos A. Coello Coello, and Luis V. Santana-Quintero. EMOPSO: A multi-objective particle swarm optimizer with emphasis on efficiency. In *Proceedings of the Fourth International Conference on Evolutionary Multi-Criterion Optimization (EMO 2007)*, Lecture Notes in Computer Science, pages 272–285. Springer, 2007.

[PE03] Ulrich Paquet and Andries P. Engelbrecht. A new particle swarm optimiser for linearly constrained optimisation. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, volume 1, pages 227–233, 2003.

[PKB07] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization – an overview. *Swarm Intelligence*, 1(1):33–57, 2007.

[PL07] Riccardo Poli and William B. Langdon. Markov chain models of bare-bones particle swarm optimizers. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO07)*, pages 142–149. ACM Press, 2007.

[PLCS07] Riccardo Poli, William B. Langdon, Maurice Clerc, and Christopher R. Stephens. Continuous optimization theroy made easy? Finite-element models of evolutionary strategies, genetic algorithms and particle swarm optimizers. In *Foundations of Genetic Algorithms*, 2007.

[Pol08] Riccardo Poli. Dynamics and stability of the sampling distribution of particle swarm optimisers via moment analysis. *Journal of Artificial Evolution and Applications*, 8(2):Article No. 15, 10 pages, 2008.

[PR97] Scott Pierce and David Rosen. NURBS-based variational modeling as a tool for the analysis of geometric tolerances. In *Proceedings of the 1997 ASME Design Engineering Technical Conference*, 1997. 10 pages, CD-ROM Proceedings.

[PSL05] Kenneth Price, Rainer Storn, and Jouni Lampinen. *Differential evolution – a practical approach to global optimization*. Springer, 2005.

[PV02a] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method in multiobjective problems. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 603–607. ACM Press, 2002.

[PV02b] Konstantinos E. Parsopoulos and Michael N. Vrahatis. Particle swarm optimization method for constrained optimization problems. In P. Sincak, J. Vascak, V. Kvasnicka, and J. Pospichal, editors, *Intelligent Technologies – Theory and Application: New Trends in Intelligent Technologies*, volume 76 of *Frontiers in Artificial Intelligence and Applications*, pages 214–220. IOS Press, 2002.

[R D08] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008.

[Rec65] Ingo Rechenberg. Cybernetic solution path of an experimental problem (1965). In David B. Fogel, editor, *Evolutionary Computation – The Fossil Record*. IEEE Press, 1998, 1965.

[Rec73] Ingo Rechenberg. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog Verlag, 1973.

[Rec94] Ingo Rechenberg. *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik*. Frommann-Holzboog Verlag, 1994.

[Ree83] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2(2):91–108, 1983.

[Rey87] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[RHW04] Asanga Ratnaweera, Saman K. Halgamuge, and Harry C. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 8(3):240–255, 2004.

[RHW10] Thomas Ritscher, Sabine Helwig, and Rolf Wanka. Design and experimental evaluation of multiple adaptation layers in self-optimizing particle swarm optimization. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010)*, 2010. To appear.

[RN88] Joseph Lee Rodgers and W. Alan Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988.

[RN03] Stuart Russell and Peter Norvig. *Artificial intelligence – A modern approach*. Pearson Education, 2nd edition, 2003.

[RRS04] Jacob Robinson and Yahya Rahmat-Samii. Particle swarm optimization in electromagnetics. *IEEE Transactions on Antennas and Propagation*, 52(2):397–407, 2004.

[RSC06] Margarita Reyes-Sierra and Carlos A. Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 2(3):287–308, 2006.

[RV03] Mark Richards and Dan Ventura. Dynamic sociometry in particle swarm optimization. In *Proceedings of the Sixth International Conference on Computational Intelligence and Natural Computing*, pages 1557–1560, 2003.

[RV04] Mark Richards and Dan Ventura. Choosing a starting configuration for particle swarm optimization. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 2309–2312, 2004.

[SC05] Margarita Reyes Sierra and Carlos A. Coello Coello. Improving PSO-based multi-objective optimization using crowding, mutation and ε-dominance. In *Evolutionlary Multi-Criterion Optimization*, pages 505–519. Springer, 2005.

[Sch75] H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung*. Dissertation, TU Berlin, Germany, 1975.

[SE98] Yuhui Shi and Russell C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 69–73, 1998.

[SE99] Yuhui Shi and Russell C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, volume 3, pages 1946–1950, 1999.

[SHL$^+$05] Ponnuthurai N. Suganthan, Nikolaus Hansen, Jing J. Liang, Kalyanmoy Deb, Y.P. Chen, Anne Auger, and S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. KanGAL Report 2005005, Nanyang Technological University, Singapore, 2005.

[SK06] B. Suman and P. Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(10):1143–1160, 2006.

[SP97] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[ST94] Rajneet Sodhi and Joshua U. Turner. Relative positioning of variational part models for design analysis. *Computer-Aided Design*, 26(5):366–378, 1994.

[Sto06] Tobias Stoll. Generieren von nichtidealer Geometrie. In *Proceedings of the 17. Symposium Design for X*, pages 143–150, 2006.

[Sug99] P. N. Suganthan. Particle swarm optimiser with neighbourhood operator. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 3, pages 1958–1962. IEEE Press, 1999.

[SW08] Dirk Sudholt and Carsten Witt. Runtime analysis of binary PSO. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 135–142. ACM Press, 2008.

[SWHP07] Tobias Stoll, Stefan Wittmann, Sabine Helwig, and Kristin Paetzold. Registration of measured and simulated non-ideal geometry using optimization methods. In *Proceedings of the 10th CIRP International Seminar on Computer Aided Tolerancing*. Chair Quality Management and Manufacturing Metrology, University of Erlangen-Nuremberg, 2007. 10 pages, CD-ROM Proceedings.

[SWM09] Tobias Stoll, Stefan Wittmann, and Harald Meerkamm. Tolerance analysis with detailed part modeling including shape deviations. In *Proceedings of the 11th CIRP International Seminar on Computer Aided Tolerancing*, 2009. 5 pages, CD-ROM Proceedings.

[Tre03] Ioan Cristian Trelea. The particle swarm optimization algorithm: Convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325, 2003.

[TS06] Tetsuyuki Takahama and Setsuko Sakai. Solving constrained optimization problems by the ε constrained particle swarm optimizer with adaptive velocity limit control. In *Proceedings of the 2006 IEEE International Conference on Cybernetics and Intelligent Systems*, pages 1–7, 2006.

[Tuf07] Edward R. Tufte. *The visual display of quantitative information*. Graphics Press LLC, 2nd edition, 2007.

[Tur90] Joshua U. Turner. Relative positioning of parts in assemblies using mathematical programming. *Computer-Aided Design*, 22(7):394–400, 1990.

[vdB02] Frans van den Bergh. *An analysis of particle swarm optimizers*. PhD thesis, University of Pretoria, 2002.

[vdBE06] F. van den Bergh and A.P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937–971, 2006.

[VF05] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. *Computational Intelligence and Bioinspired Systems*, pages 758–770, 2005.

[VL00] David A. Van Veldhuizen and Gary B. Lamont. On measuring multi-objective evolutionary algorithm performance. In *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, volume 1, pages 204–211, 2000.

[vLA87] Peter J. M. van Laarhoven and Emile H. L. Aarts. *Simulated annealing: Theory and applications*. D. Reidel Publishing Company, 1987.

[VOK07] Kalyan Veeramachaneni, Lisa Osadciw, and Ganapathi Kamath. Probabilistically driven particle swarms for optimization of multi valued discrete problems: Design and analysis. In *Proceedings of the 2007 IEEE Swarm Intelligence Symposium*, pages 141–149, 2007.

[Wil45] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[Win92] Patrick Henry Winston. *Artificial intelligence*. Addison-Wesley Publishing Company, 3rd edition, 1992.

[Wit09] Carsten Witt. Why standard particle swarm optimizers elude a theoretical runtime analysis. In *Proceedings of the tenth ACM SIGEVO Workshop on Foundations of Genetic Algorithms (FOGA'09)*, pages 13–20, 2009.

[WM97] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[WSP07] Stefan Wittmann, Tobias Stoll, and Kristin Paetzold. Volume visualization of geometric deviations. In *Proceedings of the International Conference on Engineering Design*, 2007. 6 pages, CD-ROM Proceedings.

[WSZ$^+$04] Mark P. Wachowiak, Renata Smolíková, Yufeng Zheng, Jacek M. Zurada, and Adel S. Elmaghraby. An approach to multimodal biomedical image registration utilizing particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):289–301, 2004.

[YII03] Keiichiro Yasuda, Azuma Ide, and Nobuhiro Iwasaki. Adaptive particle swarm optimization. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 1554–1559, 2003.

[YKF$^+$00] Hirotaka Yoshida, Kenichi Kawata, Yoshikazu Fukuyama, Shinichi Takayama, and Yosuke Nakanishi. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE Transactions on Power Systems*, 15(4):1232–1239, 2000.

[ZDT00] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[ZF03] Barbara Zitová and Jan Flusser. Image registration methods: A survey. *Image and Vision Computing*, 21(11):977–1000, 2003.

[ZK04] Eckart Zitzler and Simon Künzli. Indicator-based selection in multiobjective search. In *Proceedings of Parallel Problem Solving from Nature – PPSN VIII*, pages 832–842. Springer, 2004.

[ZLT01] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Swiss Federal Institute of Technology (ETH) Zurich, May 2001.

[ZT98] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms – a comparative case study. In *Parallel Problem Solving from Nature – PPSN V*, pages 292–301. Springer, 1998.

[ZT99] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[ZTL$^+$03] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions Evolutionary Computation*, 7(2):117–132, 2003.

[ZWLK09] Karin Zielinski, Petra Weitkemper, Rainer Laur, and Karl-Dirk Kammeyer. Optimization of power allocation for interference cancellation with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 13(1):128–150, 2009.

[ZXB04] Wen-Jun Zhang, Xiao-Feng Xie, and De-Chun Bi. Handling boundary constraints for numerical optimization by particle swarm flying in periodic search space. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, volume 2, pages 2307–2311, 2004.