

STEP-OPTIMIZED PARTICLE SWARM OPTIMIZATION

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Thomas Schoene

©Thomas Schoene, August 2011. All rights reserved.

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

ABSTRACT

Particle swarm optimization (PSO) is widely used in industrial and academic research to solve optimization problems. Recent developments of PSO show a direction towards adaptive PSO (APSO). APSO changes its behaviour during the optimization process based on information gathered at each iteration. It has been shown that APSO is able to solve a wide range of difficult optimization problems efficiently and effectively. In classical PSO, all parameters are fixed for the entire swarm. In particular, all particles share the same settings of their velocity weights. We propose four APSO variants in which every particle has its own velocity weights. We use PSO to optimize the settings of the velocity weights of every particle at every iteration, thereby creating a step-optimized PSO (SOPSO). We implement four known PSO variants (global best PSO, decreasing weight PSO, time-varying acceleration coefficients PSO, and guaranteed convergence PSO) and four proposed APSO variants (SOPSO, moving bounds SOPSO, repulsive SOPSO, and moving bound repulsive SOPSO) in a PSO software package. The PSO software package is used to compare the performance of the PSO and APSO variants on 22 benchmark problems. Test results show that the proposed APSO variants outperform the known PSO variants on difficult optimization problems that require large numbers of function evaluations for their solution. This suggests that the SOPSO strategy of optimizing the settings of the velocity weights of every particle improves the robustness and performance of PSO.

ACKNOWLEDGEMENTS

I would like to thank my supervisors Dr. Simone A. Ludwig and Dr. Raymond J. Spiteri for their excellent and persistent support. I would further like to thank my thesis committee Dr. Anh V. Dinh, Dr. Mark G. Eramian, and Dr. Ian McQuillan for their time and expert comments. Further, I would like to thank my colleagues from the Numerical Simulation and Intelligent Systems labs, notably Andrew Kroshko and Mahsa Naseri. Additionally, I would like to thank Jan Thompson and Gwen Lancaster for all the organizational support during my study. I would like to thank the Department of Computer Science and all its members for their support. In particular, I would like to thank Dr. Simone A. Ludwig, Dr. Raymond J. Spiteri, and the Department of Computer Science for their financial support that made this thesis possible.

I would like to dedicate this thesis to my parents Harald and Christa Schöne.

CONTENTS

Permission to Use	i
Abstract	ii
Acknowledgements	iii
Contents	v
List of Tables	vii
List of Figures	ix
List of Abbreviations	x
1 Introduction	1
1.1 Optimization	1
1.2 Optimization Problems	2
1.2.1 Types of Optimization Problems	3
1.2.2 Examples of Optimization Problems	5
1.3 Summary of Results	5
1.4 Thesis Outline	6
2 Related Work	7
2.1 Optimization Techniques	7
2.2 Particle Swarm Optimization	7
2.2.1 Global Best Particle Swarm Optimization	8
2.2.2 Decreasing Weight Particle Swarm Optimization	9
2.2.3 Time-Varying Acceleration Coefficients PSO	10
2.2.4 Guaranteed Convergence Particle Swarm Optimization	10
2.2.5 Other Particle Swarm Optimization Variants	11
2.2.6 Hybrid Particle Swarm Optimization	13
2.2.7 Adaptive Particle Swarm Optimization	13
3 Approaches and Implementation	16
3.1 Adaptive Particle Swarm Optimization	16
3.1.1 Inspiration	16
3.1.2 Step-Optimized Particle Swarm Optimization	17
3.1.3 Moving Bound Step-Optimized Particle Swarm Optimization	20
3.1.4 Repulsive Step-Optimized Particle Swarm Optimization	25
3.1.5 Moving Bound Repulsive Step-Optimized PSO	27
3.2 Standard Settings and Implementation Details	29
3.2.1 Global Best Particle Swarm Optimization	29
3.2.2 Decreasing Weight Particle Swarm Optimization	31
3.2.3 Time-Varying Acceleration Coefficients PSO	32
3.2.4 Guaranteed Convergence Particle Swarm Optimization	33
3.2.5 Step-Optimized Particle Swarm Optimization	33
3.2.6 Moving Bound Step-Optimized Particle Swarm Optimization	33
3.2.7 Repulsive Step-Optimized Particle Swarm Optimization	35
3.2.8 Moving Bound Repulsive Step-Optimized PSO	35
3.3 Software Package	36

4 Experiments	38
4.1 Test Functions	38
4.2 Experimental Setup	46
4.3 Results	47
4.3.1 Summary of Results	61
5 Conclusion	62
5.1 Contributions	62
5.2 Further Work	63
References	64
A PSO Software Package Parameters	69
B Sample Call to PSO Software Package	74
C Test Function Configurations	77
D Result Tables considering Average Solutions	78
E Comparison considering Harder Problems	83

LIST OF TABLES

2.1 Four-state APSO: change of c_1 and c_2 depending on the state.	14
3.1 GBPSO Settings	32
3.2 DWPSO Settings	32
3.3 TVACPSO Settings	32
3.4 GCPSO Settings	33
3.5 SOPSO Settings	34
3.6 MSOPSO Settings	35
3.7 RSOPSO Settings	36
3.8 MRSOPSO Settings	37
4.1 Count of wins, draws, and losses for 7,500,000 FE considering minimum solutions. . .	48
4.2 Count of wins, draws, and losses for 1,500,000 FE considering minimum solutions. . .	48
4.3 Count of wins, draws, and losses for 250,000 FE considering minimum solutions. . .	49
4.4 Count of wins, draws, and losses for 3,000 FE considering minimum solutions. . . .	49
4.5 PSO variants vs. APSO variants for 7,500,000 FE considering minimum solution. . .	50
4.6 PSO variants vs. APSO variants for 1,500,000 FE considering minimum solution. . .	50
4.7 PSO variants vs. APSO variants for 250,000 FE considering minimum solution. . . .	51
4.8 PSO variants vs. APSO variants for 3,000 FE considering minimum solution.	52
4.9 Proposed APSO comparison for 7,500,000 FE considering minimum solutions.	52
4.10 Proposed APSO comparison for 1,500,000 FE considering minimum solutions.	52
4.11 Proposed APSO comparison for 250,000 FE considering minimum solutions.	53
4.12 Proposed APSO comparison for 3,000 FE considering minimum solutions.	54
4.13 PSO comparison for 7,500,000 FE considering minimum solutions.	55
4.14 PSO comparison for 1,500,000 FE considering minimum solutions.	55
4.15 PSO comparison for 250,000 FE considering minimum solutions.	56
4.16 PSO comparison for 3,000 FE considering minimum solutions.	57
D.1 Count of wins, draws, and losses for 7,500,000 FE considering average solutions. . . .	78
D.2 Count of wins, draws, and losses for 1,500,000 FE considering average solutions. . . .	78
D.3 Count of wins, draws, and losses for 250,000 FE considering average solutions.	78
D.4 Count of wins, draws, and losses for 3,000 FE considering average solutions.	78
D.5 PSO variants vs. APSO variants for 7,500,000 FE considering average solutions. . . .	78
D.6 PSO variants vs. APSO variants for 1,500,000 FE considering average solutions. . . .	79
D.7 PSO variants vs. APSO variants for 250,000 FE considering average solutions. . . .	79
D.8 PSO variants vs. APSO variants for 3,000 FE considering average solutions.	79
D.9 Proposed APSO comparison for 7,500,000 FE considering average solutions.	80
D.10 Proposed APSO comparison for 1,500,000 FE considering average solutions.	80
D.11 Proposed APSO comparison for 250,000 FE considering average solutions.	80
D.12 Proposed APSO comparison for 3,000 FE considering average solutions.	81
D.13 PSO comparison for 7,500,000 FE considering average solutions.	81
D.14 PSO comparison for 1,500,000 FE considering average solutions.	81
D.15 PSO comparison for 250,000 FE considering average solutions.	82
D.16 PSO comparison for 3,000 FE considering average solutions.	82
E.1 PSO vs. APSO for 15,000,000 FE, minimum solution, and increased dimensionality. . .	83
E.2 PSO vs. APSO for 7,500,000 FE, minimum solution, and increased dimensionality. . .	83
E.3 PSO vs. APSO for 250,000 FE, minimum solution, and increased dimensionality. . . .	83
E.4 PSO vs. APSO for 15,000,000 FE, average solution, and increased dimensionality. . .	84
E.5 PSO vs. APSO for 7,500,000 FE, average solution, and increased dimensionality. . . .	84

E.6	PSO vs. APSO for 250,000 FE, average solution, and increased dimensionality. . . .	84
-----	--	----

LIST OF FIGURES

1.1	Example of a unimodal objective function.	3
1.2	Example of a multimodal objective function with three local minima.	4
2.1	Flowchart of PSO.	9
3.1	Flowchart of SOPSO.	20
3.2	Flowchart of MSOPSO.	24
3.3	State diagram of RSOPSO.	26
3.4	Flowchart of RSOPSO.	28
3.5	Flowchart of MRSOPSO.	30
4.1	The topography of Ackley, Equation (4.1), for two dimensions.	39
4.2	The topography of Alpine, Equation (4.2), for two dimensions.	40
4.3	The topography of Parabola, Equation (4.14), for two dimensions.	44
4.4	The topography of Rastrigin, Equation (4.15), for two dimensions.	45
4.5	The topography of Rosenbrock, Equation (4.16), for two dimensions.	46
4.6	GCPSO vs. SOPSO on Rastrigin considering minimum solutions.	58
4.7	GBPSO vs. SOPSO on Rosenbrock considering minimum solutions.	58
4.8	DWPSO vs. SOPSO on Non-continuous Rastrigin considering minimum solutions.	59
4.9	TVACPSO vs. SOPSO on Michalewicz considering average solutions.	59
4.10	DWPSO vs. SOPSO on Non-continuous Rastrigin considering average solutions.	60
4.11	DWPSO vs. RSOPSO on Rastrigin considering average solutions.	60
4.12	GBPSO vs. SOPSO on Rosenbrock considering average solutions.	61

LIST OF ABBREVIATIONS

APSO	adaptive particle swarm optimization
DWPSO	decreasing weight particle swarm optimization
FE	number of function evaluations
GBPSO	global best particle swarm optimization
GCPSO	guaranteed convergence particle swarm optimization
MRSOPSO	moving bound repulsive step-optimized particle swarm optimization
MSOPSO	moving bound step-optimized particle swarm optimization
PSO	particle swarm optimization
RSOPSO	repulsive step-optimized particle swarm optimization
SOPSO	step-optimized particle swarm optimization
TVACPSO	time-varying acceleration coefficients particle swarm optimization

CHAPTER 1

INTRODUCTION

Optimization problems are problems for which a solution, for example the highest yield or the lowest cost, is to be found. There are many different types of optimization problems. Optimization techniques often focus on specific types of optimization problems, whereas this work aims to develop optimization techniques that efficiently and effectively solve a wide range of optimization problems. *Particle swarm optimization* (PSO) [28] is a promising technique from the field of evolutionary computation that, by its nature, is able to solve a wide variety of optimization problems. We propose variants to PSO to increase its performance and its ability to solve a wide range of optimization problems. We particularly focus on *adaptive PSO* (APSO), i.e., a variant of PSO that can change its behavior based on information gained during the optimization. We present promising results, that suggest the proposed APSO variants can solve an even wider range of difficult problems efficiently and effectively.

1.1 Optimization

Optimization is the process used to find the best solution that addresses a certain problem. This research focuses on techniques from *evolutionary computation*, a subfield of heuristic and meta-heuristic optimization techniques based on successful concepts and patterns found in evolution and nature [13]. Examples of successful optimization concepts in nature are selective breeding, swarms following successful individuals, ant colonies trying to maximize their food sources by protecting plant louse colonies, or birds trying to minimize the distance to their food sources by strategically placing their nests.

This work focuses on the optimization technique PSO [28]. The swarm of PSO can be envisioned as multiple birds (particles) that search for the best food source (optimum) by using their inertia, their knowledge, and the knowledge of the swarm. Single particles behave similarly because traditionally they share the same configuration, i.e., the same settings for their velocity weights; see below. PSO is selected for the study in this thesis due to its wide use in fields like engineering [13], its ability to solve hard problems [23], its potential for use as a hybrid incorporating other optimization techniques [46, 61], and its potential for parallelization [30, 44, 58].

APSO variants adapt their behavior based on their state or knowledge gained about the objective function. Further, APSO variants have shown promising results [7, 15, 21, 72, 74, 76, 80]. Based on these reasons, this work focuses on APSO and proposes new APSO variants. The idea of the proposed APSO is to assign every particle its own velocity weights. These velocity weights are optimized during the solution process. Additionally, we combine the proposed APSO approach with ideas from attractive-repulsive PSO to create further APSO variants. We compare the proposed APSO variants to other PSO variants and show that the proposed APSO variants have improved performance on difficult problems that require large numbers of function evaluations for their solution.

1.2 Optimization Problems

An important part of an optimization problem is the function to be optimized. This function is called the *objective function* and is denoted by f in this work. In the literature, the objective function is also referred to as the fitness function. The variables required by the objective function are called *input variables*. In the literature, the input variables are also referred to as the decision variables. A particular setting of the input variables is referred to as the *position* and is denoted by \mathbf{x} , $\mathbf{x} \in \mathbb{R}^D$, where D is the total number of input variables; the literature additionally uses the terms setting or decision vector. Variables can be constrained by simple bounds or potentially complex constraints; the set of all feasible positions of the constrained variables is referred to as the *search space*. In the literature, the search space is also referred to as the function space. A *neighborhood* of a position is a connected subset of the search space containing that position. The result to which the objective function evaluates a certain position is called the *objective value*. In the literature, the objective value is also referred to as the fitness value, the fitness score, or the solution value. The objective function, its search space, and its constraints are all parts of the *optimization problem*.

The *maximum* of an objective function is the objective value where the objective function has its highest objective value. The *minimum* of an objective function is the objective value where the objective function has its lowest objective value. Figure 1.1 shows the unimodal objective function $f(x) = x^2 + 0.5$ with a minimum 0.5 at $x = 0$. Both a maximum and a minimum can be called an *optimum*. A *local optimum* is an optimum within its neighborhood. Figure 1.2 shows the multimodal objective function $f(x) = 10 \sin(x) + x$ and three of its local minima. The *global optimum* is the optimum of all local optima; i.e., it is the optimum in the complete search space. Figure 1.2 shows a global minimum of roughly -18 located close to $x = -8$ in the search space $[-10, 10]$.

A *maximization problem* is an optimization problem for which the position with the highest

objective value is to be found. A *minimization problem* is an optimization problem for which the position with the lowest objective value is to be found. A maximization problem can be converted into a minimization problem by negating the objective function. Accordingly, only minimization problems are discussed in this work. Depending on their properties, optimization problems can be divided into several types, which we now discuss.

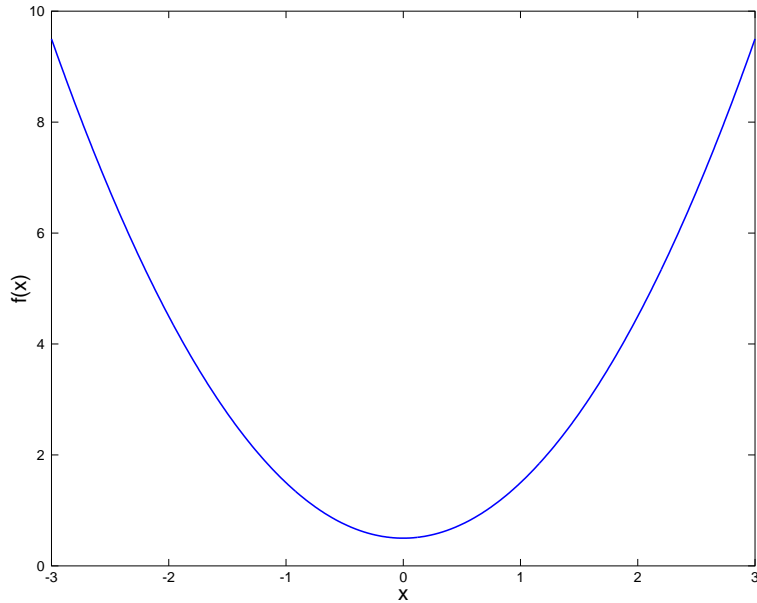


Figure 1.1: Example of a unimodal objective function.

1.2.1 Types of Optimization Problems

There are many optimization problem types and their descriptions can overlap [23]. A *convex* optimization problem has a convex objective function and convex constraints. Geometrically, convex means that a line drawn from any position z to any other position w in the search space lies on or above all objective values between z and w . Figure 1.1 shows such a convex function. *Linear programming* problems are a subgroup of convex optimization problems. A linear programming problem has a linear objective function and all constraints are linear functions. A linear function has the form $f(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2 + \dots + a_Dx_D$.

A *smooth non-linear* optimization problem has a smooth non-linear objective function or one or more smooth non-linear constraints. Smooth means that the function has derivatives at all orders. Smooth non-linear optimization problems can be convex or non-convex. An example of a smooth non-linear function is $f(\mathbf{x}) = x_1^3 - x_2^2 + 5$. *Quadratic programming* problems are a subset of smooth

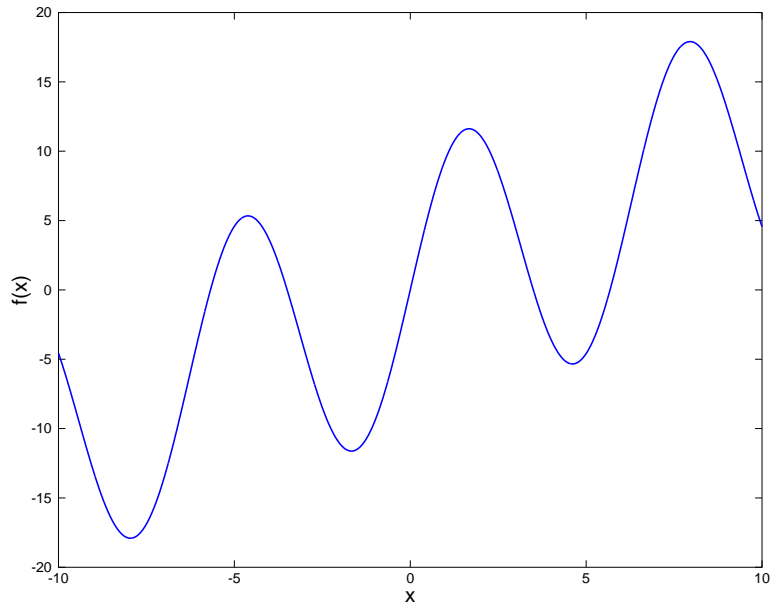


Figure 1.2: Example of a multimodal objective function with three local minima.

non-linear optimization problems. A quadratic programming problem has a quadratic objective function and linear constraints. An example of a quadratic function is $f(\mathbf{x}) = 20x_1^2 + 5x_2^2 + x_3 - 10$. Quadratic programming problems can be convex or non-convex [23].

Mixed-integer programming problems have fully or partially discrete positions. For *combinatorial* optimization problems, discrete values have to be assigned. An example of a combinatorial optimization problem is to assign consumers to limited supplies. Combinatorial optimization problems can have complex constraints. Most mixed-integer and combinatorial optimization problems are non-convex, making them hard to solve [23].

Global optimization problems are problems for which the global optimum is to be found [50]. A typical global optimization problem can be envisioned as a mountain range where the highest mountain for maximization or the lowest valley for minimization is to be found. Due to the possibility of multiple local optima, global optimization problems are often hard to solve. Like global optimization problems, *non-smooth* optimization problems generally have multiple local optima [23]. Additionally, non-smooth optimizations problems are discontinuous in some cases [23].

Dynamic optimization problems have objective functions that change over time [21, 67], making them difficult to solve. *Dynamic dimension* optimization problems have positions with a non-constant number of dimensions, and the dimensionality of the optimum is also unknown [74].

A *multi-objective* optimization problem has multiple objectives; i.e., its objective function returns a vector of objective values. In many cases, for such problems, the *Pareto front*, which consists of all non-dominated solutions, is to be found. A solution is said to be non-dominated if there is no other solution that has better values in all its objectives. Because finding all non-dominated solutions is generally more difficult than finding one solution, multi-objective optimization problems are generally more difficult to solve than single-objective optimization problems [14, 54].

1.2.2 Examples of Optimization Problems

Objective functions often attempt to model real entities. Creating an objective function that behaves like the real entity can be a challenging task on its own. Simplified descriptions of real-world optimization problems include:

- maximize the volume of a structure given a certain amount of building material [68]
- minimize the air resistance of a car body [42]
- minimize the output of certain chemical species by finding an optimal reaction temperature and pressure given a certain catalyst [20, 37]
- minimize the building cost of a car, ship, engine, or notebook without violating quality constraints [10, 18]
- minimize the length of a route that visits certain points at least once [26]
- maximize the potential yield or minimize the risk of a portfolio [4]
- maximize the efficiency of a fuel cell [6]
- minimize the difference between a simulation and experimental measurements [63]
- minimize the operating cost of a fresh water system without violating constraints such as minimal amount of stored water [79]
- minimize the difference between power generation and demand for scheduling a hydroelectric power station [7, 36, 43]

1.3 Summary of Results

The proposed APSO variants show promising results on a test suite of 22 benchmark optimization problems. For large numbers of function evaluations, the proposed APSO variants outperform implemented known PSO variants. This suggests that the proposed APSO variants improves the efficiency and effectiveness of PSO.

1.4 Thesis Outline

The outline of this thesis is as follows. Chapter 2 describes known PSO variants used in this research and various other PSO variants. Chapter 3 focuses on the proposed APSO variants, implementation details, standard settings, and their implementation in a PSO software package. Chapter 4 contains a comparison of the proposed APSO variants against known PSO variants. Chapter 5 contains conclusions and further work.

CHAPTER 2

RELATED WORK

This chapter outlines different global optimization techniques in Section 2.1 and lists PSO variants in Section 2.2. The PSO variants described include standard PSO, hybrid PSO, and APSO.

2.1 Optimization Techniques

Many optimization techniques exist. There are deterministic optimization techniques such as interval optimization [5], branch-and-bound [66], and algebraic techniques [65]. Further, there are stochastic optimization techniques such as simulated annealing [16], Monte Carlo sampling [12], stochastic tunneling [34], and parallel tempering [38]. Furthermore, there are heuristic and meta-heuristic optimization techniques such as genetic algorithms [13], evolutionary strategies [47], evolutionary programming [1], PSO [28], ant colony optimization [71], cuckoo search [75], and memetic algorithms [9]. By focusing on statistical considerations, the no-free-lunch theorem describes a set of theorems that state that there is no perfect optimization technique [69]. If there is an optimization technique “A” that outperforms optimization technique “B” on optimization problem “X”, there is an optimization problem “Y” for which optimization technique “B” outperforms optimization technique “A”. The no-free-lunch theorem further states that if optimization technique “C” is specialized to certain problems, it can be expected to show good performance for these problems, but it can be expected to show weak performance for other problems. The no-free-lunch theorem also discusses that creating a general optimization technique that performs well on many optimization problems is challenging and that such a general optimization technique is likely to be outperformed on a given optimization problem by specialized solvers, i.e., a better solution is found using the same number of function evaluations.

2.2 Particle Swarm Optimization

PSO was introduced by Kennedy and Eberhart [28]. The behavior of PSO can be envisioned by comparing it to bird swarms searching for optimal food sources, where the direction in which a bird moves is influenced by its current movement, the best food source it ever experienced, and the

best food source any bird in the swarm ever experienced. In other words, birds are driven by their inertia, their personal knowledge, and the knowledge of the swarm. In terms of PSO, the movement of a particle is influenced by its inertia, its personal best position, and the global best position.

PSO has multiple particles, and every particle consists of its current objective value, its position, its velocity, its *personal best value*, that is the best objective value the particle ever experienced, and its *personal best position*, that is the position at which the personal best value has been found. In addition, PSO maintains the *global best value*, that is the best objective value any particle has ever experienced, and the *global best position*, that is the position at which the global best value has been found. Classical PSO [28] uses the following iteration to move the particles:

$$\mathbf{x}^{(i)}(n+1) = \mathbf{x}^{(i)}(n) + \mathbf{v}^{(i)}(n+1), \quad n = 0, 1, 2, \dots, N-1, \quad (2.1a)$$

where $\mathbf{x}^{(i)}$ is the position of particle i , n is the iteration, $n = 0$ refers to the initialization, N is the total number of iterations, and $\mathbf{v}^{(i)}$ is the velocity of particle i . In classical PSO, the velocity of the particle is determined using the following iteration:

$$\begin{aligned} \mathbf{v}^{(i)}(n+1) &= \mathbf{v}^{(i)}(n) + 2\mathbf{r}_1^{(i)}(n)[\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)] + 2\mathbf{r}_2^{(i)}(n)[\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)], \\ n &= 0, 1, 2, \dots, N-1, \end{aligned} \quad (2.1b)$$

where \mathbf{x}_p is the personal best position, and \mathbf{x}_g is the global best position. $\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)$ calculates a vector directed towards the personal best position, and $\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)$ calculates a vector directed towards the global best position. Both $\mathbf{r}_1^{(i)}$ and $\mathbf{r}_2^{(i)}$ are random vectors that contain values uniformly distributed between 0 and 1. The notation $\mathbf{r}_1^{(i)}(n)$ is meant to denote that a new random vector is generated for every particle i and iteration n .

PSO can focus on either convergence or diversity at any iteration. To focus on diversity means particles are scattered, searching a large area coarsely. To focus on convergence means particles are close to each other, searching a small area intensively. A promising strategy is to focus on diversity in early iterations and convergence in later iterations [13, 55].

2.2.1 Global Best Particle Swarm Optimization

Global best PSO (GBPSO) [13] is a standard PSO variant. GBPSO [28] uses the following iteration to determine the velocities:

$$\begin{aligned} \mathbf{v}^{(i)}(n+1) &= w\mathbf{v}^{(i)}(n) + c_1\mathbf{r}_1^{(i)}(n)[\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)] + c_2\mathbf{r}_2^{(i)}(n)[\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)], \\ n &= 0, 1, 2, \dots, N-1. \end{aligned} \quad (2.2)$$

The velocity weights are the inertia weight w , the personal best weight c_1 , and the global best weight c_2 . Compared to the classical PSO described in [28], GBPSO uses an inertia weight w and does not require the personal c_1 and global best weight c_2 be set to 2 as in Equation (2.1b).

Figure 2.1 shows the flowchart of GBPSO. First, the swarm is initialized; i.e., the position and velocity of particles are randomly initialized within the search space. After that, the objective values of particles are calculated. The first objective values and positions are automatically the personal best values and the personal best positions. The global best value and the global best position are set to the objective value and position of the particle with the best objective value in the entire swarm. After the initial step, all particles are moved to their new positions using Equation (2.1a). All objective values are evaluated again. Personal best positions are updated for particles that have a new objective value that is better than the old personal best value. The global best position is updated if there is any particle with an objective value that is better than the old global best value. Again, all particles are moved to their new positions using Equation (2.1a). The algorithm continues with evaluating the objective values and updating the positions, the personal best values, the personal best positions, the global best value, and the global best position. The algorithm stops if a termination criterion, such as a limit on the number of iterations, is reached.

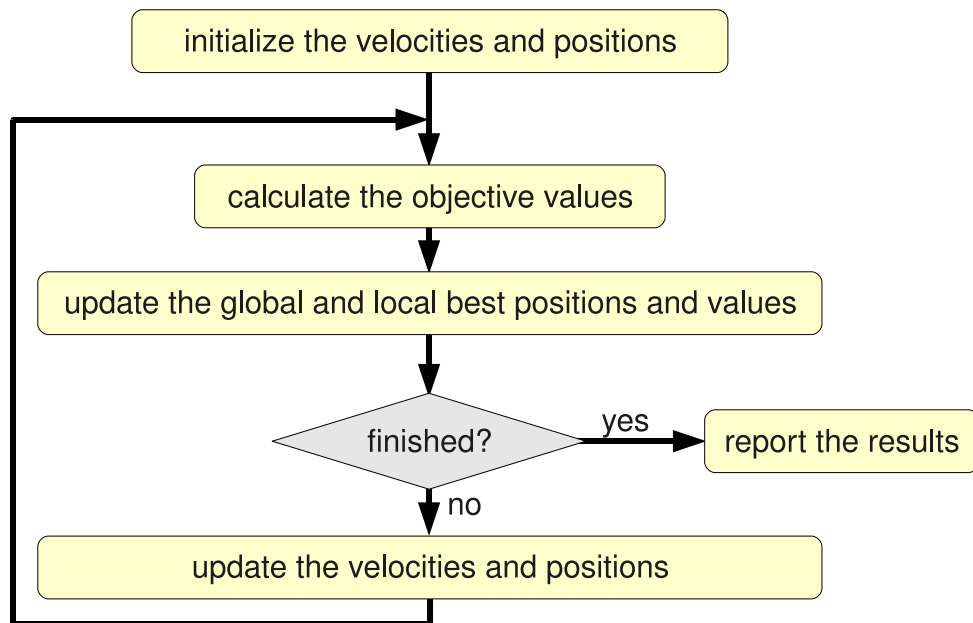


Figure 2.1: Flowchart of PSO.

2.2.2 Decreasing Weight Particle Swarm Optimization

Decreasing weight particle swarm optimization (DWPSO) is similar to GBPSO, but the inertia weight is decreased linearly over time [13]. The idea behind DWPSO is to focus on diversity in early iterations and convergence in late iterations. DWPSO uses the following iteration to determine

the velocities:

$$\begin{aligned}\mathbf{v}^{(i)}(n+1) &= w(n)\mathbf{v}^{(i)}(n) + c_1\mathbf{r}_1^{(i)}(n)[\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)] + c_2\mathbf{r}_2^{(i)}(n)[\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)], \\ n &= 0, 1, 2, \dots, N-1,\end{aligned}\tag{2.3}$$

where the inertia weight w at every iteration n is calculated using the following equation:

$$w(n) = w_s - (w_s - w_e)\frac{n}{N},\tag{2.4}$$

where $w(n)$ is the inertia weight at iteration n , w_s is the inertia weight designated for the first iteration, and w_e is the inertia weight designated for the last iteration N .

The inertia weight w at every iteration n depends on the total number of iterations N . Therefore, changing the total number of iterations N will change the behavior of the algorithm at every iteration. This also means that the algorithm cannot be restarted from a certain point if the total number of iterations N is changed.

2.2.3 Time-Varying Acceleration Coefficients PSO

Time-varying Acceleration Coefficients PSO (TVACPSO) does not only change the inertia weight w , but also the acceleration coefficients, i.e., the personal c_1 and global best weight c_2 , over time [13, 55]. The idea is to have a high diversity for early iterations and a high convergence for late iterations. The inertia weight w is changed as in DWPSO using Equation (2.4). TVACPSO uses the following iteration to determine the velocities:

$$\begin{aligned}\mathbf{v}^{(i)}(n+1) &= w(n)\mathbf{v}^{(i)}(n) + c_1(n)\mathbf{r}_1^{(i)}(n)[\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)] + c_2(n)\mathbf{r}_2^{(i)}(n)[\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)], \\ n &= 0, 1, 2, \dots, N-1,\end{aligned}\tag{2.5}$$

where the personal best weight c_1 and the global best weight c_2 at every iteration n are calculated using the following equations:

$$c_1(n) = c_{1s} - (c_{1s} - c_{1e})\frac{n}{N},\tag{2.6a}$$

$$c_2(n) = c_{2s} - (c_{2s} - c_{2e})\frac{n}{N},\tag{2.6b}$$

where $c_1(n)$ is the personal best weight at iteration n , $c_2(n)$ is the global best weight at iteration n , c_{1s} is the personal best weight designated for the first iteration, c_{1e} is the personal best weight designated for the last iteration N , c_{2s} is the global best weight designated for the first iteration, and c_{2e} is the global best weight designated for the last iteration N .

2.2.4 Guaranteed Convergence Particle Swarm Optimization

Guaranteed Convergence PSO (GCPSO) guarantees that the global best particle searches within a dynamically adapted radius at all times [64]. The technique addresses the problem of stagnation

and increases local convergence [64] by using the global best particle to randomly search in an adaptively changing radius at every iteration. GCPSO, as described in [64], uses Equation (2.3) to determine the velocities $\mathbf{v}^{(i)}(n)$ and Equation (2.4) to change the inertia weight w over time. The personal best weight c_1 and the global best weight c_2 are kept constant. GCPSO uses the following iteration to update the position of the global best particle:

$$\begin{aligned} \mathbf{v}^{(i_g)}(n+1) &= -\mathbf{x}^{(i_g)}(n) + \mathbf{x}_g(n) + w(n)\mathbf{v}^{(i_g)}(n) + \rho(n)(1 - 2\mathbf{r}_3(n)), \\ n &= 0, 1, 2, \dots, N-1, \end{aligned} \quad (2.7a)$$

where i_g is the index of the particle that updated the global best value most recently. The expression $-\mathbf{x}^{(i_g)}(n) + \mathbf{x}_g(n)$ is used to reset the position of particle i_g to the global best position. The random numbers in $\mathbf{r}_3(n)$ are uniformly distributed between 0 and 1. The search radius is controlled by the search radius parameter ρ . The search radius parameter ρ is calculated using:

$$\rho(n+1) = \begin{cases} 2\rho(n), & \text{if } \check{s}(n+1) > s_c, \\ \frac{1}{2}\rho(n), & \text{if } \check{a}(n+1) > a_c, \\ \rho(n), & \text{otherwise,} \end{cases} \quad (2.7b)$$

where s_c is the success threshold and a_c is the failure threshold. Success means using Equation (2.7a) results in an improved global best value and position, and failure means it does not. The numbers of consecutive successes $\check{s}(n)$ and failures $\check{a}(n)$ are calculated using:

$$\check{s}(n+1) = \begin{cases} 0, & \text{if } \check{a}(n+1) > \check{a}(n), \\ \check{s}(n) + 1, & \text{otherwise,} \end{cases} \quad (2.7c)$$

$$\check{a}(n+1) = \begin{cases} 0, & \text{if } \check{s}(n+1) > \check{s}(n), \\ \check{a}(n) + 1, & \text{otherwise.} \end{cases} \quad (2.7d)$$

2.2.5 Other Particle Swarm Optimization Variants

Local best PSO differs from GBPSO in that the global best position \mathbf{x}_g used in Equation (2.1b) is replaced with the global best position of a certain subset of particles [13]. Therefore, different particles may not use the same global best position \mathbf{x}_g . A fully informed PSO is introduced in [27]. The paper recommends to cluster particles into k clusters based on their position after initialization. The neighborhood size is increased as the algorithm proceeds, trying to achieve a totally connected swarm, i.e., one cluster, after 80% of the total number of iterations N . Local best PSO is studied by comparing the weight factor free canonical PSO and fully informed PSO for different topologies; i.e., neighborhood structures and neighborhood sizes, in [29]. Weight factor free canonical PSO does not use any weights; i.e., no inertia weight, no personal best weight, and no global best weight.

The paper shows that large neighborhoods contribute to convergence, whereas small neighborhoods contribute to diversity [29].

Mutation TVACPSO combines time-varying acceleration coefficients, time-varying inertia weight, mutation, maximum velocity, and reinitialization of particles [55]. If any particle stagnates, it is reinitialized. If the global best particle stagnates, it is mutated. Further, one particle is selected randomly and values at random dimensions in its position and velocity are mutated.

An attractive-repulsive PSO [56] jumps between an attractive phase and a repulsive phase based on the diversity in the swarm. The attractive phase focuses on convergence, and the repulsive phase focuses on diversity. The attractive phase uses Equation (2.1b) from GBPSO to calculate the velocities $\mathbf{v}^{(i)}$. The repulsive phase uses Equation (2.1b) with a repulsive personal best $-c_1$ and a repulsive global best $-c_2$ weight to calculate the velocities. Two thresholds are used, a minimal diversity threshold for which the algorithm switches to the repulsive phase, and a maximal diversity threshold for which the algorithm switches to the attractive phase.

The cooperative combinatorial PSO [31] is specialized for solving mixed-integer and combinatorial optimization problems. Cooperative combinatorial PSO uses multiple swarms to optimize different parts of the problem by moving particles towards the global best position of their swarm and the global best positions of neighboring swarms [31].

There are PSO variants specialized for solving dynamic optimization problems. Multi-swarm charged PSO and self-organizing scouts PSO were developed for solving dynamic optimization problems [45]. Multi-swarm charged PSO uses charged sub-swarms that repel each other to maintain diversity. Self-organizing scouts PSO splits the swarm if a local optimum is found, one part focusing on the local optimum and the other part on finding another optimum. The cooperative charged PSO is a combination of a cooperative PSO and a multi-swarm charged PSO [53]. Cooperative charged PSO divides the search space by dimensions, and every sub-swarm optimizes the objective values for certain dimensions. Cooperative charged PSO uses a context vector that holds the values of the global best sub-positions. Particles use the context vector to fill dimensions in the position that their sub-swarm does not optimize. Cooperative combinatorial PSO uses the concept of particles that are charged and repel each other as well as particles that are neutral and do not repel each other to increase diversity and counter dynamic changes in the objective function.

Vector evaluated PSO [48] and multi-objective APSO [25] are specialized for solving multi-objective optimization problems. Vector evaluated PSO uses one swarm for every objective. Each swarm evaluates the objective values based on the particular objective upon which it focuses and uses one of the others swarms global best particles as its global best particle [45].

Optimized PSO uses PSO to find good parameter settings for the whole swarm [40]. Optimized PSO runs multiple PSO instances and uses the optima from the individual runs to optimize the parameter settings [40].

2.2.6 Hybrid Particle Swarm Optimization

Hybrid optimization techniques combine two or more optimization techniques. A hybrid PSO technique combines PSO with one or more optimization techniques.

The memetic PSO combines linear search strategies with PSO [59]. Similarly, fuzzy adaptive Nelder–Mead PSO uses fuzzy adaptive PSO as a global search technique and applies a Nelder–Mead simplex search to the global best position after every iteration [27].

The co-evolutionary PSO [61] uses multiple competing swarms. Objective values are calculated using the objective function and the global best value of a competing swarm. Contrary to co-evolutionary PSO, the multi-swarm co-evolution PSO [77] uses cooperating swarms. Multi-swarm co-evolution PSO uses one master swarm that contains the global best particles of all swarms. One half of the particles of all slave swarms move towards their personal best position, the global best position, and the global best position of the master swarm. The other half uses crossover and mutation. Particles in the master swarm move towards the global best position and the global best position of the previous iteration.

2.2.7 Adaptive Particle Swarm Optimization

A promising research direction within PSO is the use of adaptive concepts and patterns [7, 15, 21, 72, 74, 76, 80]. APSO variants adapt their behavior based on information gathered as the optimization proceeds.

There are many APSO variants that adaptively change some or all of the velocity weights w , c_1 , and c_2 . PSO with dynamic adaption [76] uses an evolutionary speed factor that measures personal best value changes and an aggregation degree that measures the relative position of particles in the objective space to calculate the inertia weight w . The APSO in [82] adapts the inertia weight of every particle based on its objective value, the global best value, and the global worst value. The APSO introduced in [2] changes its inertia weight based on swarm diversity to reduce premature convergence and hence increases overall convergence. The swarm diversity is calculated as a function of positions. Different variations of the self-tuning APSO are discussed in [72, 73, 78]. Self-tuning APSO as described in [72] grants every particle its own personal best weight $c_1^{(i)}$ and global best weight $c_2^{(i)}$. Self-tuning APSO initializes the personal and global best weights randomly for every particle and then moves them towards the values of the particle that yielded the most updates of the global best position, where the magnitude of the movement is based on the total number of iterations [73]. In an update of self-tuning APSO, the personal and global best weights are moved in ever smaller steps for increasing iterations [72]. In a recent update of self-tuning APSO, the authors show how to self-adapt the inertia weights $w^{(i)}$, the personal best weights $c_1^{(i)}$, and the global best weights $c_2^{(i)}$ [78]. The controlled APSO adapts the distance of single particles to the global best

particle to address the problems of stagnating particles, exponentially increasing velocity sizes, and trapping of particles at local optima [22]. A parameter is calculated to adapt the personal best weights $c_1^{(i)}$ and the global best weights $c_2^{(i)}$. The inertia weight APSO [82] grants every particle its own inertia weight $w^{(i)}$ and adapts the inertia weights as a function of the objective values and the global best objective value. The self-learning APSO adapts personal best weights $c_1^{(i)}$ of particles based on environment feedback, i.e., feedback given by the objective function [3]. The personal best weights $c_1^{(i)}$ are updated using a function of that environment feedback [3].

The APSO introduced in [24] adaptively sets the velocity weights w , c_1 , and c_2 of the swarm based on the average velocity. The four-state APSO remains in one of four states and changes its personal best weight c_1 and the global best weight c_2 based on the current state [80]. The four states used are exploration, exploitation, convergence, and jumping-out. The state is selected for every iteration using a fuzzy set on an evolutionary factor. The evolutionary factor is calculated using the mean separation of the particles of the swarm. Table 2.1 shows the changes in the personal best weight c_1 and the global best weight c_2 based on the state.

Table 2.1: Four-state APSO: change of c_1 and c_2 depending on the state.

State	c_1	c_2
Exploration	Increase	Decrease
Exploitation	Slight increase	Slight decrease
Convergence	Slight increase	Slight increase
Jumping-out	Decrease	Increase

A two-state APSO is introduced in [32]. Two-state APSO avoids being trapped at local optima and the failure to accurately find the global optima by putting particles in either an explorative or an exploitative state. In the explorative state, the particles use Equation (2.1b) for updating the position. In the exploitative state, the particles are driven away from their personal best and worst positions. The state of a particle is determined by comparing the distance between its position and the global best position to an activity threshold that is decreased over time.

Certain APSO variants adapt entities such as the random distributions used. Fuzzy APSO uses a scaled random velocity to speed up particles that have a velocity $\mathbf{v}^{(i)}$ smaller than the minimum velocity threshold \mathbf{v}_{\min} [35]. This speed up counters decreasing velocities that might hold the algorithm in a local optimum. Fuzzy APSO calculates the scale factor and \mathbf{v}_{\min} using a fuzzy set and expert rules.

Mutation is used by several APSO variants. In PSO with adaptive mutation [15], mutation is used to increase diversity and to reduce the chance of premature convergence. The personal

best position of every particle is mutated based on the variance of the objective values and the aggregation degree of the particles, i.e., the relative positions of the particles in the solution space. Mutation is used by [62] to avoid early convergence. The global best particle is changed with a certain probability, which is determined by the variance in the objective values. Low variance leads to many mutations, whereas high variance leads to few mutations.

Detection and response APSO is used to solve dynamic optimization problems by monitoring the global best and second best positions [21]. Detection and response APSO assumes that a change in the objective function occurred if any of these two positions change their objective value between iterations. If the objective function changes, particles and the global best particle are reinitialized at random positions.

Dimension adaptive PSO addresses dynamic dimension optimization problems by using a probability equation to calculate the number of dimensions for positions [74]. An adaptive vector is used to determine which dimension is added or deleted in case the number of dimensions of a position changes.

Multi-objective APSO addresses highly constrained multi-objective problems by using a gradient-based search on non-dominated particles [25]. The weight of the gradient-based technique is adaptively changed based on the distribution of non-dominated particles.

CHAPTER 3

APPROACHES AND IMPLEMENTATION

This chapter describes the proposed APSO variants in Section 3.1, states standard settings and implementation details for the implemented known PSO and proposed APSO variants in Section 3.2, and introduces the implemented PSO software package in Section 3.3.

3.1 Adaptive Particle Swarm Optimization

As part of this work, different APSO variants have been implemented. Step-optimized PSO (SOPSO) is described in Section 3.1.2, moving bound SOPSO (MSOPSO) is described in Section 3.1.3, repulsive SOPSO (RSOPSO) is described in Section 3.1.4, and moving bound repulsive SOPSO (MRSOPSO) is described in Section 3.1.5.

3.1.1 Inspiration

The idea for our novel concept is inspired by PSO techniques that give every particle its own velocity weights [72, 78]. These techniques often move the velocity weights of all particles to the velocity weights of a certain particle [78] that is selected based on superior performance. Self-tuning APSO moves the velocity weights towards the settings of the particle that yielded the most updates of the global best position [72, 78]. Controlled APSO adaptively changes the personal best weights $c_1^{(i)}$ and the global best weights $c_2^{(i)}$ based on the distance between the positions and the global best position [22]. Inertia weight APSO [82] grants every particle its own inertia weight $w^{(i)}$ that is changed using a function of the objective values and the global best objective value. Inspired by optimized PSO [40], we go further by treating the problem of finding good velocity weights as another optimization problem within the original optimization problem. Optimized PSO uses multiple PSO subswarms, each with its own parameter settings, in an inner iteration to solve the original optimization problem. The parameter settings are then optimized in an outer iteration of PSO for a fixed number of iterations. A detailed description of optimized PSO including a flow diagram can be found in [40].

3.1.2 Step-Optimized Particle Swarm Optimization

In step-optimized PSO (SOPSO) every particle has its own velocity weights, i.e., its inertia weight $w^{(i)}$, its personal best weight $c_1^{(i)}$, and its global best weight $c_2^{(i)}$. As mentioned above, the proposed SOPSO treats the problem of finding good velocity weights for every particle as an optimization problem. A particular setting of the velocity weights is referred to as the *position of the velocity weights*. An objective function for the velocity weights is used to calculate how well the positions of the velocity weights perform for solving the optimization problem. Using the calculated objective values of the velocity weights, SOPSO optimizes the velocity weights using a PSO variant such as GBPSO, TVACPSO, or DWPSO at every iteration. The velocity weights are optimized in a fixed search space for the velocity weights.

Compared to optimized PSO [40], as described above, the SOPSO approach of optimizing after every iteration is more efficient because we only run one PSO instance for exactly one iteration a total number of $N - 1$ times. Another advantage of the proposed SOPSO approach is that the velocity weights can adapt themselves to dynamic changes, i.e., different conditions, for example different distributions of the particles, at different iterations.

SOPSO allows the selection of certain particles for mutation (see Table 3.5); mutation is used in other APSO variants such as [62]. If a particle is selected for mutation, its velocity weights are reinitialized at every iteration. This mutation is used to maintain diverse behavior of particles and to better address the dynamic optimization problem of optimizing the velocity weights. The problem of optimizing the velocity weights is a dynamic optimization problem because a good setting for a certain iteration and situation, for example, requiring velocity weights that yield diversity might be a bad setting for another iteration and situation, for example requiring velocity weights that yield convergence.

SOPSO maintains the global best value, the global best position, the global best objective value of the velocity weights, the global best position of the velocity weights, and the particles, where every particle stores its current objective value, current position, personal best value, personal best position, velocity weights, objective value of the velocity weights, personal best objective value of the velocity weights, and personal best position of the velocity weights.

SOPSO uses the following iterations with the notation as used in Equations (2.1a) and (2.2) to update the velocities of particles:

$$\begin{aligned} \mathbf{v}^{(i)}(n+1) &= w^{(i)}(n)\mathbf{v}^{(i)}(n) + c_1^{(i)}(n)\mathbf{r}_1^{(i)}(n)[\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)] + c_2^{(i)}(n)\mathbf{r}_2^{(i)}(n)[\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)], \\ n &= 0, 1, 2, \dots, N - 1, \end{aligned} \tag{3.1}$$

where $w^{(i)}$ are the inertia weights, $c_1^{(i)}$ are the personal best weights, and $c_2^{(i)}$ are the global best weights of particle i .

The objective function for the velocity weights is used to calculate the success of particles.

Whereas the success of particles depends on their velocity weights, there are directly employable, more accurate, and reliable entities to measure the performance of particles. In particular, we use the improvement in the objective value of the particle, the number of updates of the global best position the particle yielded, and the number of updates of the personal best position the particle yielded to measure the success of particles. We propose the following objective function for the velocity weights, selected based on good performance in tests:

$$\tilde{f}^{(i)}(n) = e^{(i)}(n)(1 + \check{l}u_p^{(i)}(n) + \check{g}u_g^{(i)}(n)), \quad n = 1, 2, \dots, N - 1, \quad (3.2)$$

where $\tilde{f}^{(i)}(n)$ is the objective value of the velocity weights for particle i at iteration n , e is the normalized improvement, $u_p^{(i)}$ is the number of times particle i updated its local best position, $u_g^{(i)}$ is the number of times particle i updated the global best position, \check{l} is the local weight factor used to weigh the number of local best updates $u_p^{(i)}$, and \check{g} is the global weight factor used to weigh the number of global best updates $u_g^{(i)}$. The value of \check{g} is usually set to a higher number as the value of \check{l} as updates to the global best position are more important. By using the objective value of the velocity weights \tilde{f} , the personal best and the global best position of the velocity weights can be found. Alternative objective functions are for example to only use the normalized improvements $e^{(i)}$ or the local and global best update counters. The normalized improvements $e^{(i)}$ are calculated using the following equation, selected based on good performance in tests:

$$e^{(i)}(n) = \frac{\delta^{(i)}(n)}{\check{s}(n)}, \quad (3.3a)$$

where $\check{s}(n)$ is the normalization sum (described below) and $\delta^{(i)}(n)$ is the difference in the objective values calculated using:

$$\delta^{(i)}(n) = f^{(i)}(n) - f^{(i)}(n - 1), \quad (3.3b)$$

where $f^{(i)}$ is the objective value of particle i .

In practice, early iterations might yield large absolute values of $\delta^{(i)}$, whereas late iterations might only yield small absolute values of $\delta^{(i)}$. Therefore, we propose the following normalization to give good positions of the velocity weights from late iterations a chance of being selected as local or global best positions of the velocity weights. In other words, the normalization sum $\check{s}(n)$ makes objective values of the velocity weights from different iterations n comparable. The normalization sum $\check{s}(n)$ is calculated as follows:

$$\check{s}(n) = \begin{cases} \sum_{i=1}^{n_p} \max(-\delta^{(i)}(n), 0), & \text{if any } \delta^{(i)}(n) < 0, \\ 1, & \text{otherwise,} \end{cases} \quad (3.3c)$$

where n_p is the number of particles in the swarm. This normalization is selected based on good performance in tests.

The velocity weights are optimized using PSO. The following iterations are applied to update the positions of the velocity weights:

$$\tilde{\mathbf{x}}^{(i)}(n+1) = \tilde{\mathbf{x}}^{(i)}(n) + \tilde{\mathbf{v}}^{(i)}(n+1), \quad n = 1, 2, 3, \dots, N-1, \quad (3.4a)$$

$$\begin{aligned} \tilde{\mathbf{v}}^{(i)}(n+1) = & \tilde{w}(n)\tilde{\mathbf{v}}^{(i)}(n) + \tilde{c}_1(n)\tilde{\mathbf{r}}_1^{(i)}(n)[\tilde{\mathbf{x}}_p^{(i)}(n) - \tilde{\mathbf{x}}^{(i)}(n)] \\ & + \tilde{c}_2(n)\tilde{\mathbf{r}}_2^{(i)}(n)[\tilde{\mathbf{x}}_g^{(i)}(n) - \tilde{\mathbf{x}}^{(i)}(n)] \quad , n = 1, 2, \dots, N-1, \end{aligned} \quad (3.4b)$$

where $\tilde{\mathbf{x}}$ is the position of the velocity weights, $\tilde{\mathbf{v}}$ is the velocity of the velocity weights, $\tilde{\mathbf{x}}_p$ is the personal best position of the velocity weights, $\tilde{\mathbf{x}}_g$ is the global best position of the velocity weights, \tilde{w} is the inertia weight for optimizing the velocity weights, \tilde{c}_1 is the personal best weight for optimizing the velocity weights, \tilde{c}_2 is the global best weight for optimizing the velocity weights, and $\tilde{\mathbf{r}}_1^{(i)}$ and $\tilde{\mathbf{r}}_2^{(i)}$ are random vectors. The components in the random vectors $\tilde{\mathbf{r}}_1^{(i)}$ and $\tilde{\mathbf{r}}_2^{(i)}$ are uniformly distributed between 0 and 1 for every particle i and iteration n . Equations (3.4) are used after Equation (2.1a) has been used to update the positions of the particles and the new objective values have been calculated. The first component of $\tilde{\mathbf{x}}^{(i)}$ is used as the inertia weight $w^{(i)}$, the second component of $\tilde{\mathbf{x}}^{(i)}$ is used as the personal best weight $c_1^{(i)}$, and third component of $\tilde{\mathbf{x}}^{(i)}$ is used as the global best weight $c_2^{(i)}$ in Equation (3.1).

Figure 3.1 shows the flowchart of SOPSO. Steps that have a pale yellow background are also part of PSO (Figure 2.1) and steps that have a yellow background are introduced for SOPSO. First, the swarm is initialized; i.e., the position of particles, the velocity of particles, and positions of the velocity weights are randomly initialized within their search spaces. The objective values of the particles are calculated using the objective function. If the termination criterion is not fulfilled, the algorithm continues with moving all particles to their new positions using Equation (3.1). All objective values are evaluated again. If there are particles with objective values better than their personal best value, their personal best positions and values are updated and their local best update counter $u_p^{(i)}$ is increased by 1. The global best update counter $u_g^{(i)}$ is increased for the particle that updated the global best position and value if there is such a particle. After that, the objective values of the velocity weights are calculated for all particles using Equation (3.2). The first objective values and positions of the velocity weights are automatically the personal best values and positions of the velocity weights. The global best position and value of the velocity weights are selected by finding the particle with the best objective value of the velocity weights. The positions of the velocity weights are updated using Equations (3.4). The termination criterion is checked and if it is still not satisfied, the algorithm continues with updating the positions, evaluating the objective values, updating the personal best values and positions, updating the local best update counters, updating the global best position and value, and updating the global best update counter. This time, personal best positions and values of the velocity weights are only updated if the new objective values of the velocity weights are better. The global best position and value of the velocity

weights are updated if the particle with the best objective value of the velocity weights in the swarm has a better value than the global best objective value of the velocity weights. Again, the positions of the velocity weights are updated using Equations (3.4). The algorithm stops if a termination criterion is reached.

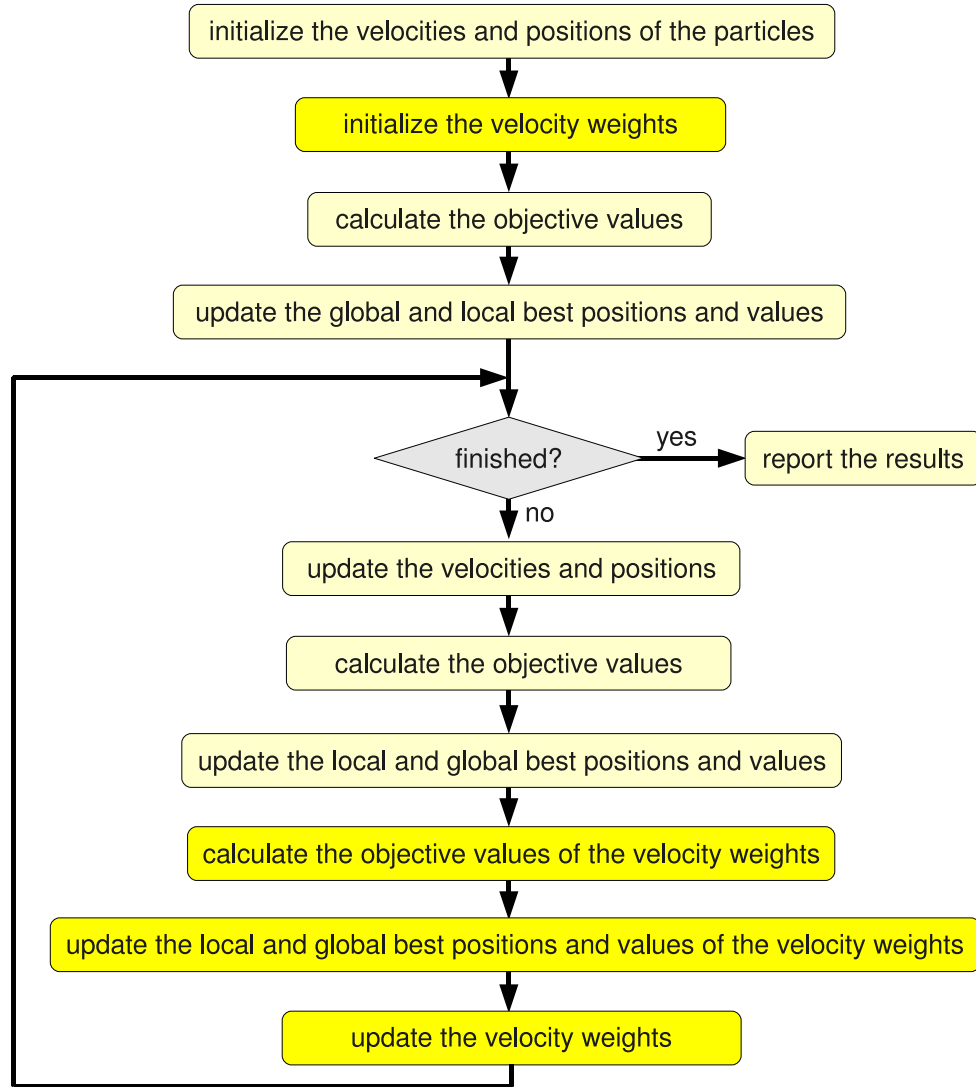


Figure 3.1: Flowchart of SOPSO.

3.1.3 Moving Bound Step-Optimized Particle Swarm Optimization

Moving bound SOPSO (MSOPSO) is a variant of SOPSO. MSOPSO gives users more influence on the behavior of SOPSO. MSOPSO makes it possible to use expert knowledge on PSO and the optimization problem addressed. Instead of using a large, fixed search space for the velocity

weights, MSOPSO uses a configurable search space for the velocity weights. Therefore, by using MSOPSO, concepts of PSO variants such as TVACPSO [13, 55] or four-state APSO [80] can be applied. MSOPSO makes it possible to optimize the velocity weights and at the same time roughly base them on the iteration number or the mean separation. This concept is inspired by the idea of using the mean separation to influence the velocity weights, as for example in four-state APSO [80], or to use the number of iterations to influence the velocity weights, as for example in TVACPSO or DWPSO [13, 55]. The distribution with which velocity weights are initialized or reinitialized is moved according to the search space for the velocity weights. The following equations are used because we have observed that they effectively move the bounds for optimizing the velocity weights between the absolute lower and upper boundaries.

The following equations are proposed to move the bounds for optimizing the inertia weights:

$$w_l(n) = \check{w}_l + s_w(n)(\check{w}_u - \check{w}_l) - s_w(n)\check{w}_w, \quad (3.5a)$$

$$w_u(n) = w_l(n) + \check{w}_w. \quad (3.5b)$$

The inertia weights $w^{(i)}(n)$ are optimized between the lower inertia weight bound $w_l(n)$ and the upper inertia weight bound $w_u(n)$. The absolute lower inertia weight \check{w}_l is the lowest value any lower inertia weight bound $w_l(n)$ can have and the absolute upper inertia weight \check{w}_u is the highest value any upper inertia weight $w_u(n)$ can have. The lower and upper inertia weight bounds, w_l and w_u , move based on the mean separation of the inertia weight $s_w(n)$ calculated from:

$$s_w(n) = \begin{cases} 1 - s(n), & \text{if } \check{w}_f < 0, \\ s(n), & \text{otherwise.} \end{cases} \quad (3.5c)$$

If the inertia weight bound flag \check{w}_f is negative, the search space for optimizing the inertia weights $w^{(i)}$ moves from the absolute upper inertia weight \check{w}_u towards the absolute lower inertia weight \check{w}_l for a decreasing mean separation $s(n)$. For example, a high mean separation at a certain iteration might yield inertia weights $w^{(i)}$ optimized in the range of 0.7 to 0.9, and a low mean separation at another iteration might yield inertia weights $w^{(i)}$ optimized in the range of 0.4 to 0.6. If the inertia weight bound flag is positive, the bound range for optimizing the inertia weights moves from the absolute lower inertia weight bound \check{w}_l towards the absolute upper inertia weight bound \check{w}_u for a decreasing mean separation $s(n)$.

Analogously, the following equations are used to move the bounds for optimizing the personal best weights $c_1^{(i)}$:

$$c_{1l}(n) = \check{c}_{1l} + s_{c1}(n)(\check{c}_{1u} - \check{c}_{1l}) - s_{c1}(n)\check{c}_{1w}, \quad (3.6a)$$

$$c_{1u}(n) = c_{1l}(n) + \check{c}_{1w}. \quad (3.6b)$$

The personal best weights $c_1^{(i)}(n)$ are optimized between the lower personal best weight bound $c_{1l}(n)$ and the upper personal best weight bound $c_{1u}(n)$. The absolute lower personal best weight \check{c}_{1l} and

the absolute upper personal best weight \check{c}_{1u} bound the values of the lower and upper personal best weight bounds. The lower c_{1l} and upper personal best weight bounds c_{1u} move based on the mean separation of the personal best weight $s_{c1}(n)$, calculated from:

$$s_{c1}(n) = \begin{cases} 1 - s(n), & \text{if } \check{c}_{1f} < 0, \\ s(n), & \text{otherwise.} \end{cases} \quad (3.6c)$$

If the personal best weight bound flag \check{c}_{1f} is negative, the search space for optimizing the personal best weights $c_1^{(i)}$ moves from the absolute upper personal best weight bound \check{c}_{1u} towards the absolute lower personal best weight bound \check{c}_{1l} for a decreasing mean separation $s(n)$. For example, a high mean separation at a certain iteration might yield personal best weights $c_1^{(i)}$ optimized in the range of 1.5 to 1.9, and a low mean separation at another iteration might yield personal best weights $c_1^{(i)}$ optimized in the range of 0.6 to 1.0.

The following equations are used to move the bounds for optimizing the global best weights $c_2^{(i)}$:

$$c_{2l}(n) = \check{c}_{2l} + s_{c2}(n)(\check{c}_{2u} - \check{c}_{2l}) - s_{c2}(n)\check{c}_{2w}, \quad (3.7a)$$

$$c_{2u}(n) = c_{2l}(n) + \check{c}_{2w}. \quad (3.7b)$$

The global best weights $c_2^{(i)}$ are optimized between the lower global best weight bound $c_{2l}(n)$ and the upper global best weight bound $c_{2u}(n)$. The absolute lower global best weight \check{c}_{2l} and the absolute upper global best weight \check{c}_{2u} bound the values of the lower and upper global best weight bounds. The lower c_{2l} and upper global best weight bounds c_{2u} move based on the mean separation of the global best weight $s_{c2}(n)$, calculated from:

$$s_{c2}(n) = \begin{cases} 1 - s(n), & \text{if } \check{c}_{2f} < 0, \\ s(n), & \text{otherwise.} \end{cases} \quad (3.7c)$$

If the global best weight bound flag \check{c}_{2f} is positive, the search space for optimizing the global best weights $c_2^{(i)}$ moves from the absolute lower global best weight bound \check{c}_{2l} towards the absolute upper global best weight bound \check{c}_{2u} for a decreasing mean separation $s(n)$. For example, a high mean separation at a certain iteration might yield global best weights $c_2^{(i)}$ optimized in the range of 0.8 to 1.2, and a low mean separation at another iteration might yield global best weights $c_2^{(i)}$ optimized in the range of 1.6 to 2.0.

The mean separation $s(n)$ is calculated using the following equation:

$$s(n) = \frac{1}{n_p} \sum_{i=1}^{n_p} s^{(i)}(n), \quad (3.8a)$$

where $s^{(i)}(n)$ is the mean separation of particle i from all other particles. A high value for the mean separation $s(n)$ implies particles are far scattered in the search space, and a low value for the mean

separation $s(n)$ implies the particles are condensed at a certain point in the search space. The sum over all mean separations $\sum_{i=1}^{n_p} s^{(i)}(n)$ is divided by the number of particles n_p , guaranteeing a normalized mean separation in $[0, 1]$ because all $s^{(i)}$ are in $[0, 1]$.

We use the Euclidean distance to calculate the mean separation of any particle as used in four-state APSO [80]. The only difference between the equation used by four-state APSO [80] and Equation (3.8b) is that we divide by the square root of the number of dimensions. Specifically, the mean separation of a single particle $s^{(i)}$ from all other particles is calculated using:

$$s^{(i)}(n) = \frac{1}{\sqrt{D}(n_p - 1)} \sum_{j=1}^{n_p} \sqrt{\sum_{d=1}^D \left(\frac{x_d^{(i)}(n) - x_d^{(j)}(n)}{b_d} \right)^2}, \quad (3.8b)$$

where $\mathbf{b} = \mathbf{u} - \mathbf{l}$.

The size of the search space \mathbf{b} is calculated subtracting the lower bounds of the search space \mathbf{l} from the upper bounds of the search space \mathbf{u} . The term $\frac{x_d^{(i)}(n) - x_d^{(j)}(n)}{b_d}$ calculates the separation of particles i and j normalized to the search space \mathbf{b} for component d . Because the sum over the separations from particle i to all particles is normalized by dividing it by the number of particles n_p and the square root of the number of dimensions \sqrt{D} , the mean separations $s^{(i)}$ of particles are in $[0, 1]$.

If calculating the mean separation $s(n)$ for every iteration takes too much computational time, MSOPSO offers the option to use the percentage of remaining iterations $m(n)$ instead. Using an approximation of the state of the system, instead of an actual measurement of the state of the system, might be sufficient to choose the right adaptation strategy for solving certain problems. If the percentage of remaining iterations is used, Equations (3.5), (3.6), and (3.7) replace the mean separation $s(n)$ with the percentage of remaining iterations $m(n)$. The percentage of remaining iterations $m(n)$ is calculated using the following equation:

$$m(n) = 1 - \frac{n}{N}.$$

The decreasing behavior of the percentage of remaining iterations $m(n)$ is desired because the mean separation $s(n)$ is expected to behave similarly; i.e., it is expected to decrease over time because all particles are attracted towards the global best position \mathbf{x}_g .

Figure 3.2 shows the flowchart of MSOPSO. The flow of MSOPSO is similar to the flow of SOPSO. Steps that have a yellow background are also part of SOPSO (Figure 3.1), and steps that have an orange background are introduced for MSOPSO. At every iteration n , including initialization, MSOPSO sets the search space for the velocity weights based on the mean separation or the percentage of remaining iterations.

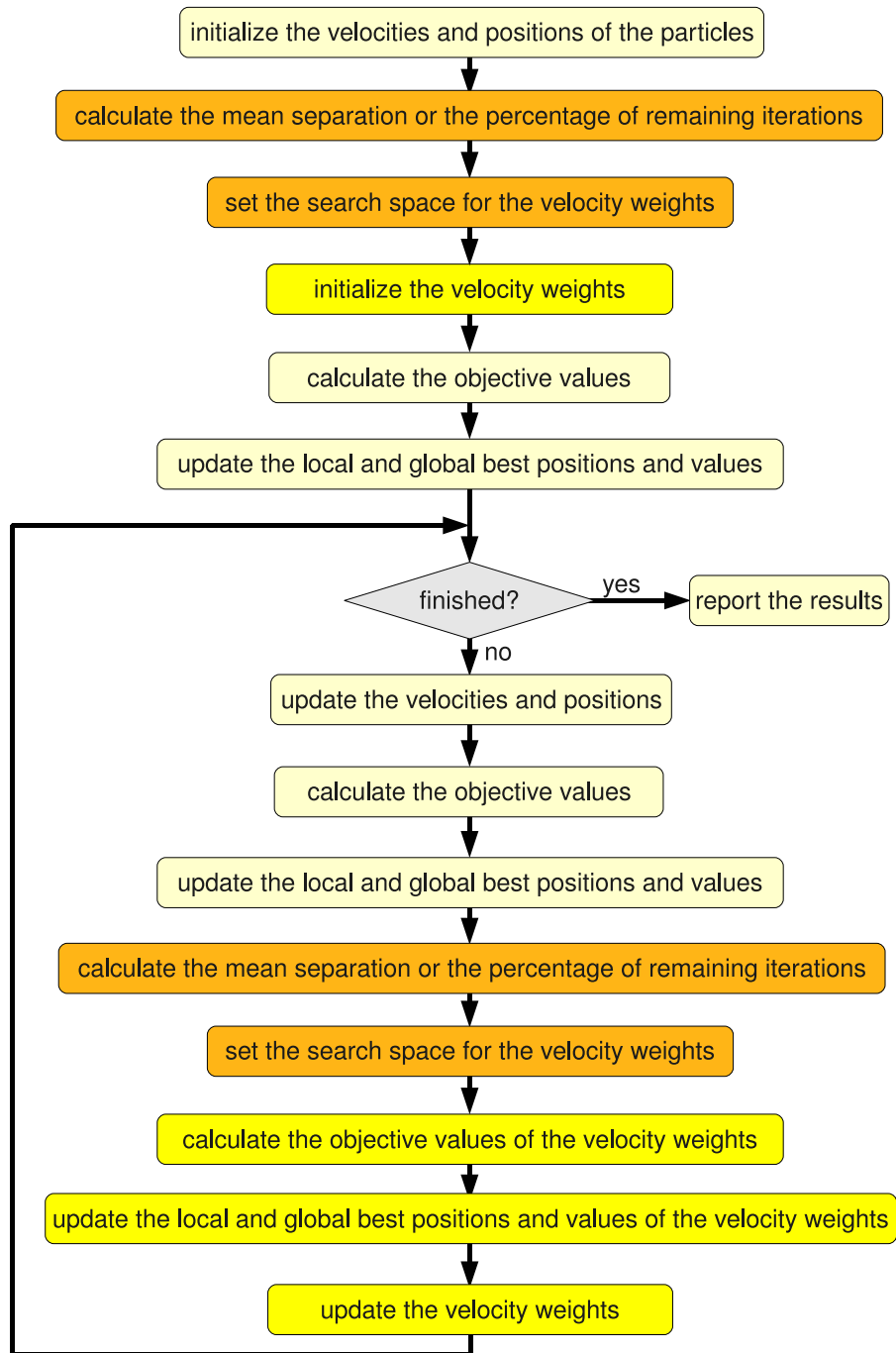


Figure 3.2: Flowchart of MSOPSO.

3.1.4 Repulsive Step-Optimized Particle Swarm Optimization

Repulsive SOPSO (RSOPSO) combines the proposed SOPSO concepts of optimizing the velocity weights $w^{(i)}$, $c_1^{(i)}$, and $c_2^{(i)}$ with an attractive and a repulsive phase as used by the attractive-repulsive PSO [56]. The proposed RSOPSO switches between phases based on the mean separation of particles. If RSOPSO is in the attractive phase and converges, it switches to the repulsive phase once it has reached a small enough mean separation. This can counter the trapping at a local optimum that might not be the global optimum. If RSOPSO is in the repulsive phase, it switches to the attractive phase once it has reached a large enough mean separation. Similarly, four-state APSO uses the mean separation to decide in which of four states it is [80]. The attractive-repulsive PSO [56] switches between phases based on a calculated diversity factor that is calculated similarly to the mean separation.

We propose the following objective function for the velocity weights that adapts itself to the current phase:

$$\bar{f}^{(i)}(n) = \begin{cases} \tilde{f}^{(i)}(n), & \text{if } a(n) = 1, \\ -s^{(i)}(n), & \text{if } a(n) = 2, \end{cases} \quad (3.9)$$

where $\bar{f}^{(i)}(n)$ is the objective value of the velocity weights used in RSOPSO and $a(n)$ is the phase indicator. If RSOPSO is in the attractive phase $a(n) = 1$, the objective value of the velocity weights $\bar{f}^{(i)}(n)$ is set to $\tilde{f}^{(i)}(n)$ as calculated in Equation (3.2). If RSOPSO is in the repulsive phase $a(n) = 2$, the objective value of the velocity weights $\bar{f}^{(i)}(n)$ is set to the negation of the mean separation $s^{(i)}(n)$ as calculated in Equation (3.8b). This objective function for the velocity weights was selected for RSOPSO because good performance of the velocity weights is indicated by $\tilde{f}^{(i)}(n)$ in the attractive phase and $-s^{(i)}(n)$ in the repulsive phase. In particular, in the attractive phase we focus on convergence by rewarding good objective values of the velocity weights $\tilde{f}^{(i)}(n)$ as in SOPSO, and in the repulsive phase we focus on diversity by rewarding high mean separations $s^{(i)}(n)$.

The attractive-repulsive PSO [56] switches to the repulsive phase if its diversity factor goes below an absolute lower threshold value and switches to the attractive phase if its diversity factor goes above an absolute upper threshold value. We use the same mechanism but replace the diversity factor with the mean separation. Specifically, we use the following equation to switch between phases:

$$a(n+1) = \begin{cases} 1, & \text{if } a(n) = 2 \wedge s(n) > s_u(n), \\ 2, & \text{if } a(n) = 1 \wedge s(n) < s_l(n), \\ a(n), & \text{otherwise,} \end{cases} \quad (3.10a)$$

where $s_l(n)$ is the mean separation absolute lower threshold and $s_u(n)$ is the mean separation

absolute upper threshold.

RSOPSO starts in the attractive phase $a(n) = 1$. If the mean separation $s(n)$ calculated by Equation (3.8a) falls below the mean separation absolute lower threshold $s_l(n)$, RSOPSO changes from the attractive phase $a(n) = 1$ to the repulsive phase $a(n+1) = 2$. If the mean separation $s(n)$ calculated by Equation (3.8a) rises above the mean separation absolute upper threshold, RSOPSO changes from the repulsive phase $a(n) = 2$ to the attractive phase $a(n+1) = 1$; this is depicted in Figure 3.3.

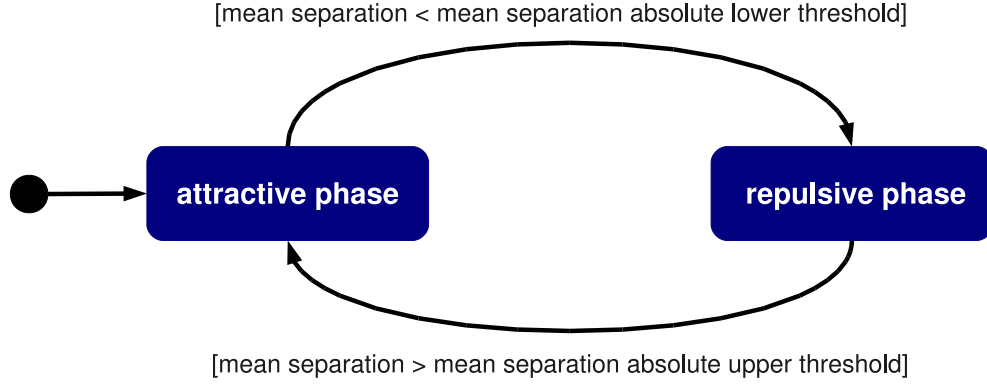


Figure 3.3: State diagram of RSOPSO.

To the best of our knowledge, the adaptive change of the mean separation absolute lower $s_l(n)$ and upper threshold $s_u(n)$ is novel. This concept allows for increased accuracy and convergence as the algorithm proceeds. Furthermore, it can be used if good values for the mean separation absolute lower and the mean separation absolute upper threshold are not known. The mean separation absolute lower $s_l(n)$ and upper threshold $s_u(n)$ are adapted as follows:

$$s_l(n+1) = \begin{cases} s_l(n)/\check{s}_l, & \text{if } a(n) = 2 \wedge s(n) > s_u(n), \\ s_l(n), & \text{otherwise,} \end{cases} \quad (3.10b)$$

$$s_u(n+1) = \begin{cases} s_u(n)/\check{s}_u, & \text{if } a(n) = 2 \wedge s(n) > s_u(n), \\ s_u(n), & \text{otherwise,} \end{cases} \quad (3.10c)$$

where \check{s}_l is the mean separation absolute lower divisor and \check{s}_u is the mean separation absolute upper divisor. The mean separation absolute lower threshold $s_l(n+1)$ is divided by the mean separation absolute lower divisor \check{s}_l and the mean separation absolute upper threshold $s_u(n+1)$ is divided by the mean separation absolute upper divisor \check{s}_u if the algorithm switches from the repulsive phase to the attractive phase at iteration n . Both the mean separation absolute lower $s_l(n+1)$ and upper threshold $s_u(n+1)$ remain the same if the algorithm does not switch from the repulsive phase to the attractive phase; i.e., the mean separation absolute lower $s_l(n+1)$ and upper threshold $s_u(n+1)$

are only changed after a full cycle through the attractive and repulsive state.

Figure 3.4 shows the flowchart of RSOPSO. The flow of RSOPSO is similar to the flow of SOPSO. Steps that have a cyan background are introduced for RSOPSO. Different from SOPSO, RSOPSO calculates the mean separation after the local and global best positions are updated. RSOPSO requires the mean separation to decide whether a phase switch is required. If so, the objective function for the velocity weights and the search space for the velocity weights are switched to their counterparts in the new phase. The search space for the velocity weights in the attractive phase must mainly yield positive velocity weights. The search space for the velocity weights in the repulsive phase must mainly yield negative velocity weights. All velocity weights have to be reinitialized in the new search space for the velocity weights if a phase switch occurred. The personal best positions and values of the velocity weights and the global best position and value of the velocity weights are reset because if their values were discovered in the attractive phase, they cannot be used in the repulsive phase and vice versa. In case a switch from the repulsive to the attractive phase occurs, i.e., one phase cycle is finished, the mean separation absolute lower and upper threshold are updated using Equations (3.10b) and (3.10c). If no phase switch occurs, RSOPSO follows the flow of SOPSO in optimizing the velocity weights; however, it uses Equation (3.9) instead of Equation (3.2) as the objective function for the velocity weights.

3.1.5 Moving Bound Repulsive Step-Optimized PSO

Moving Bound Repulsive SOPSO (MRSOPSO) is the combination of MSOPSO and RSOPSO; i.e., the search space for the velocity weights is moved as described for MSOPSO and an attractive and a repulsive phase are used as described for RSOPSO. Therefore, all novel parts of MSOPSO and RSOPSO are novel for MRSOPSO, and all known parts of MSOPSO and RSOPSO are adapted for MRSOPSO.

MRSOPSO typically uses the mean separation instead of the percentage of remaining iterations for calculating the search space for the velocity weights because the mean separation is calculated for the RSOPSO part of MRSOPSO. The mean separation used for MRSOPSO is different from the one used for MSOPSO and RSOPSO. The normalized mean separation $s(n)$ is replaced by a proposed normalized mean separation $\bar{s}(n)$ relative to the mean separation absolute lower $s_l(n)$ and upper threshold $s_u(n)$, thereby yielding values better distributed in $[0, 1]$. The normalized mean separation $\bar{s}(n)$ is calculated using the following equation:

$$\bar{s}(n) = \begin{cases} \frac{s(n)-s_l(n)}{s_u(n)-s_l(n)}, & \text{if } s_l(n) < s(n) < s_u(n), \\ 0, & \text{if } s(n) \leq s_l(n), \\ 1, & \text{if } s(n) \geq s_u(n). \end{cases} \quad (3.11)$$

MRSOPSO uses the following equations instead of Equations (3.5c), (3.6c), and (3.7c) used by

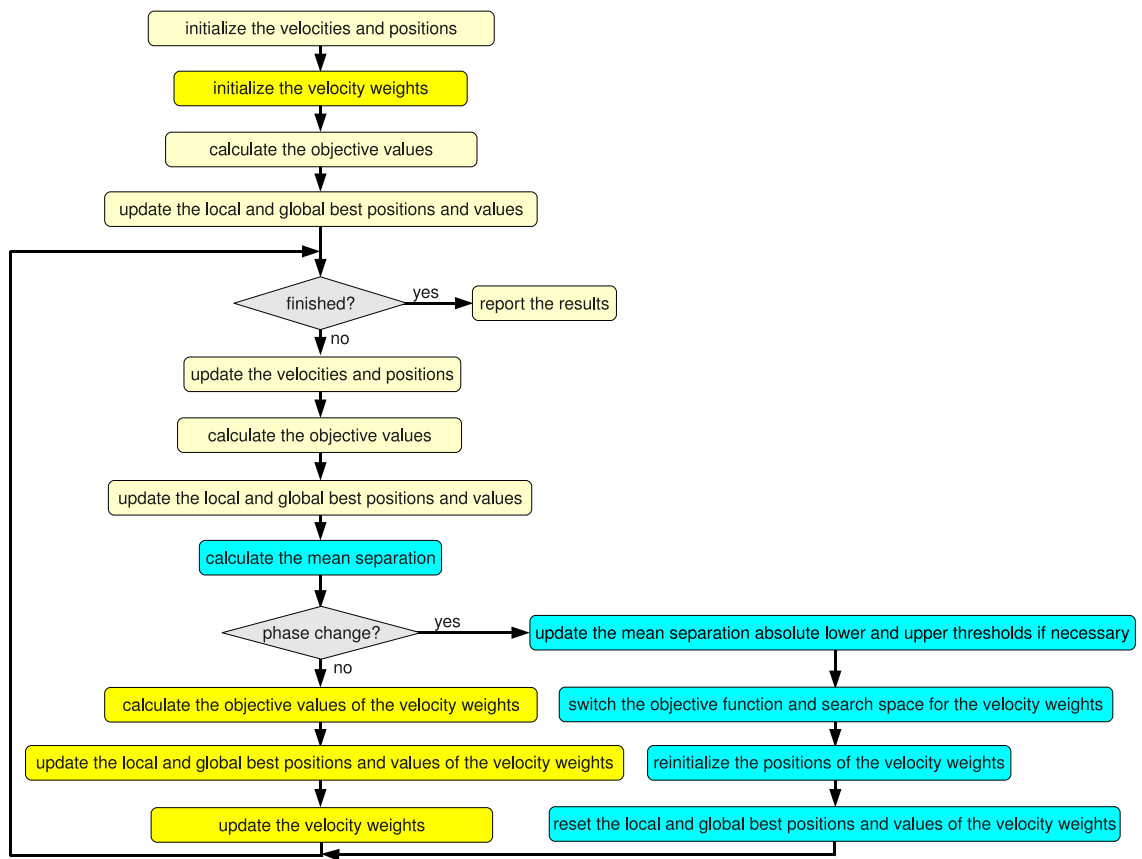


Figure 3.4: Flowchart of RSOPSO.

MSOPSO:

$$\begin{aligned}
 m_w(n) &= \begin{cases} 1 - \bar{s}(n), & \text{if } \check{w}_f < 0, \\ \bar{s}(n), & \text{otherwise,} \end{cases} \\
 m_{c1}(n) &= \begin{cases} 1 - \bar{s}(n), & \text{if } \check{c}_{1f} < 0, \\ \bar{s}(n), & \text{otherwise,} \end{cases} \\
 m_{c2}(n) &= \begin{cases} 1 - \bar{s}(n), & \text{if } \check{c}_{2f} < 0, \\ \bar{s}(n), & \text{otherwise.} \end{cases}
 \end{aligned}$$

The values of the absolute lower inertia weight \check{w}_l , the absolute upper inertia weight \check{w}_1 , the inertia weight bound width \check{w}_w , the inertia weight bound flag \check{w}_f , and accordingly \check{c}_{1l} , \check{c}_{1u} , \check{c}_{1w} , \check{c}_{1f} , \check{c}_{2l} , \check{c}_{2u} , \check{c}_{2w} , and \check{c}_{2f} are different for the repulsive and the attractive phase. Specifically, these values are set to create positive personal and global best weights in the attractive phase but negative personal and global best weights in the repulsive phase. Phase switches are detected according to Figure 3.3. Like RSOPSO, MRSOPSO uses Equation (3.9) as objective function for the velocity weights.

Figure 3.5 shows the flowchart of MRSOPSO. Steps that have an orange background are also part of MSOPSO (Figure 3.2), and steps that have a cyan background are also part of RSOPSO (Figure 3.4). The only difference with RSOPSO is that MRSOPSO calculates the mean separation to set the search space for the velocity weights before initializing the positions of the velocity weights. MRSOPSO calculates the search space for the velocity weights based on the mean separation.

3.2 Standard Settings and Implementation Details

This section focuses on the implementation details and standard settings used. If individual PSO variants share the same argument settings, they are not repeated. The parameters of SOPSO, MSOPSO, RSOPSO, and MRSOPSO have been optimized empirically and set to good, general-purpose settings; i.e., we tried different settings for every parameter and chose the best performing one. Implementation details and settings used for GBPSO are described in Section 3.2.1, for DWPSO in Section 3.2.2, for TVACPSO in Section 3.2.3, for GCPSO in Section 3.2.4, for SOPSO in Section 3.2.5, for MSOPSO in Section 3.2.6, for RSOPSO in Section 3.2.7, and for MRSOPSO in Section 3.2.8.

3.2.1 Global Best Particle Swarm Optimization

For PSO, certain aspects, such as when the algorithm is stopped or how search space violations are handled, can be implemented differently. We stop GBPSO if the iteration number n reaches the

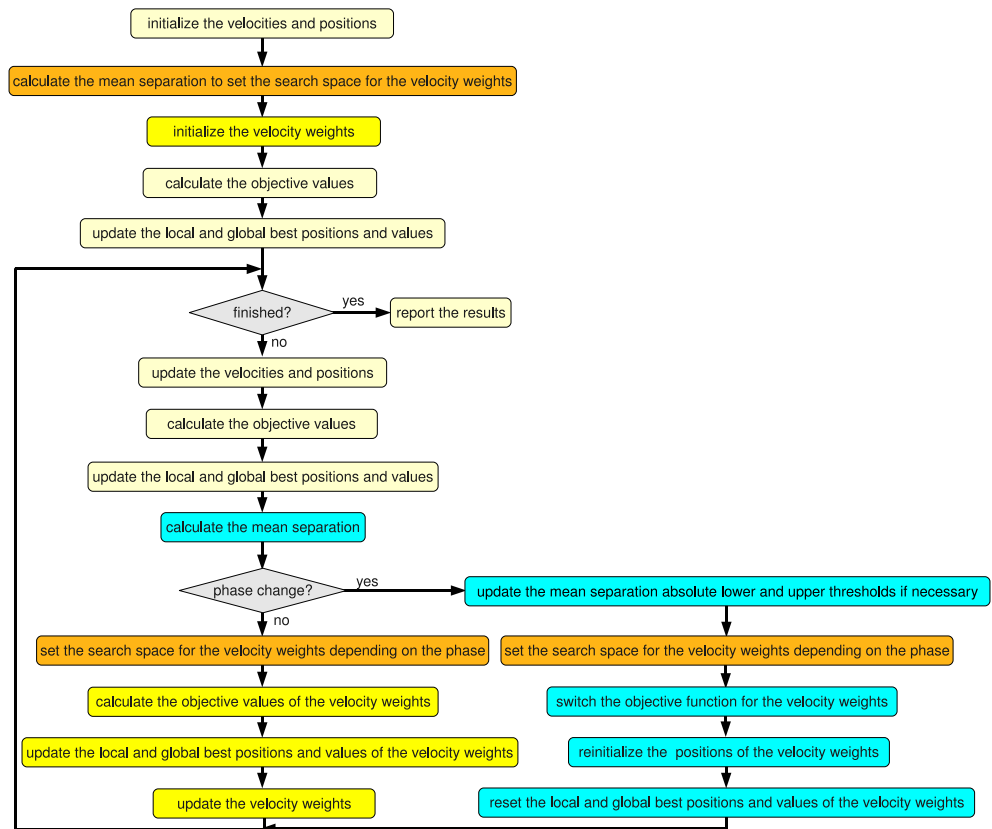


Figure 3.5: Flowchart of MRSOPSO.

total number of iterations N . If a particle leaves the search space, it is returned along the path from which it left; that is, ever smaller parts of the negated velocity are applied to the particle. In particular, we use the following iteration:

$$\check{\mathbf{x}}^{(i)}(\check{n} + 1) = \check{\mathbf{x}}^{(i)}(\check{n}) - \check{\mathbf{v}}^{(i)}(\check{n} + 1), \quad \check{n} = 0, 1, \dots, \check{N} - 1, \quad (3.12a)$$

where $\check{\mathbf{x}}^{(i)}$ is the reduced position, which will be taken as position $\mathbf{x}^{(i)}$ as soon as it is in the search space, $\check{\mathbf{v}}^{(i)}$ is the reduced velocity, \check{n} is the reducing iteration count, and \check{N} is the total number of reducing iterations. The initial reduced position $\check{\mathbf{x}}^{(i)}(0)$ is set to the position $\mathbf{x}^{(i)}(n + 1)$ that violated the search space. The reduced velocities $\check{\mathbf{v}}^{(i)}$ are calculated using:

$$\check{\mathbf{v}}^{(i)}(\check{n} + 1) = \alpha \check{\mathbf{v}}^{(i)}(\check{n}), \quad \check{n} = 0, 1, \dots, \check{N} - 1, \quad (3.12b)$$

where α is the reduction factor, and the initial reduced velocity $\check{\mathbf{v}}^{(i)}(0)$ is set to the velocity $\mathbf{v}^{(i)}(n+1)$ that caused the search space violation. Equation (3.12a) is applied starting with the first reducing iteration $\check{n} = 1$, until the reduced position $\check{\mathbf{x}}^{(i)}$ is in the search space or the limit on the total number of reducing iterations \check{N} is reached. In the unlikely case that the final reduced position $\check{\mathbf{x}}^{(i)}(\check{N})$ is not in the search space, it is reinitialized at the components for which it violates the search space. By using this technique instead of simply setting the particle to the bound for which it violated the search space, we decrease the low chance of having all positions on a bound and stopping the optimization for that dimension. In particular, we use this technique and the values chosen for the reduction factor α and for the total number of reducing iterations \check{N} based on rudimentary tests. Therefore, it might be possible to find better settings for the reduction factor α and for the total number of reducing iterations \check{N} using additional tests. Better settings for the reduction factor α and for the total number of reducing iterations \check{N} would likely increase the performance of all eight implemented PSO variants because all variants use the same search space violation handling settings.

We set the values of the inertia weight w , the personal best weight c_1 , the global best weight c_2 , the reduction factor α , and the total number of reducing iterations \check{N} according to Table 3.1. Some papers propose to set the personal best weight c_1 and the global best weight c_2 to 2 [64] but we set the personal best weight c_1 and the global best weight c_2 to 1.49618 as proposed in [52] from their observations in tests.

3.2.2 Decreasing Weight Particle Swarm Optimization

DWPSO shares many of its implementation details with GBPSO. DWPSO differs from GBPSO in that the inertia weight w is decreased for increasing iterations n using Equation (2.4). DWPSO is configured according to Table 3.2.

Table 3.1: GBPSO Settings

Parameter	Label	Setting
inertia weight	w	0.7298 [52]
personal best weight	c_1	1.49618 [52]
global best weight	c_2	1.49618 [52]
reduction factor	α	0.54
total number of reducing iterations	\check{N}	4

Table 3.2: DWPSO Settings

Parameter	Label	Setting
inertia weight designated for the first iteration	w_s	0.9 [13]
inertia weight for the last iteration	w_e	0.4 [13]
personal best weight	c_1	2 [60]
global best weight	c_2	2 [60]

3.2.3 Time-Varying Acceleration Coefficients PSO

TVACPSO shares many of its implementation details with DWPSO. TVACPSO differs from DW-PSO in that the personal best weight c_1 and global best weight c_2 are changed depending on the iteration n . TVACPSO is configured according to Table 3.3.

Table 3.3: TVACPSO Settings

Parameter	Label	Setting
inertia weight designated for first iteration	w_s	0.9 [13]
inertia weight designated for last iteration	w_e	0.4 [13]
personal best weight designated for first iteration	c_{1s}	2.5 [55]
personal best weight designated for last iteration	c_{1e}	0.5 [55]
global best weight designated for first iteration	c_{2s}	0.5 [55]
global best weight designated for last iteration	c_{2e}	2.5 [55]

3.2.4 Guaranteed Convergence Particle Swarm Optimization

GCPSO shares many of its implementation details with DWPSO. GCPSO only differs from DWPSO in that it uses Equation (2.7a) to update the position of the global best particle. GCPSO is configured according to Table 3.4.

Table 3.4: GCPSO Settings

Parameter	Label	Setting
inertia weight designated for the first iteration	w_s	0.9 [13]
inertia weight for the last iteration	w_e	0.4 [13]
personal best weight	c_1	2 [64]
global best weight	c_2	2 [64]
initial search radius argument	$\rho(0)$	1 [64]
failure threshold	a_c	5 [64]
success threshold	s_c	15 [64]

3.2.5 Step-Optimized Particle Swarm Optimization

If positions of the velocity weights leave the search space, they are moved back according to the principles used for positions that leave the search space. In particular, we use a reduction factor for the velocity weights $\tilde{\alpha}$ that can have a different value than the reduction factor α .

SOPSO can be used with different objective functions for the velocity weights; we use Equation (3.2). This concept can be helpful to evaluate what entities can be used to measure the performance of particles.

SOPSO has the option to configure the number of iterations after which the velocity weights are optimized and an option to configure the number of iterations used for optimizing the velocity weights. Both options are set to 1, meaning that Equation (3.4a) is applied exactly once for every iteration. SOPSO offers different methods for normalizing the objective values of the velocity weights; we use the normalization as in Equations (3.3). All other parameters of SOPSO are set according to Table 3.5.

3.2.6 Moving Bound Step-Optimized Particle Swarm Optimization

The implementation of MSOPSO follows the implementation of SOPSO closely. The only difference is that the search space used for optimizing the velocity weights is changed based on the mean separation during optimization. The search space for the velocity weights is used to determine the

Table 3.5: SOPSO Settings

Parameter	Label	Setting
search space for the velocity weights	-	$[-0.5, 2.0]$
search space for the personal best weights	-	$[-1.0, 4.2]$
search space for the global best weights	-	$[-1.0, 4.2]$
weight factor for the local updates	\tilde{l}	1
weight factor for the global updates	\tilde{g}	6
number of iterations after which velocity weights are optimized	-	1
number of iterations for which velocity weights are optimized	-	1
method for normalizing the objective values of the velocity weights	-	Equations (3.3)
inertia weight for optimizing velocity weights for first iteration	w_s	0.9
inertia weight for optimizing velocity weights for last iteration	w_e	0.4
personal best weight for optimizing velocity weights for first iteration	c_{1s}	2.5
personal best weight for optimizing velocity weights for last iteration	c_{1e}	0.5
global best weight for optimizing velocity weights for first iteration	c_{2s}	0.5
global best weight for optimizing velocity weights for last iteration	c_{2e}	2.5
percent of velocity weights reinitialized after every iteration	-	33%
iterations after which to reset best positions & values of velocity weights	-	50
initialization space for inertia weights	-	$[0.4, 0.9]$
initialization space for personal best weights	-	$[0.5, 2.5]$
initialization space for global best weights	-	$[0.5, 2.5]$
reinitialization space for inertia weights	-	$[0.5, 0.8]$
reinitialization space for personal best weights	-	$[0.6, 2.4]$
reinitialization space for global best weights	-	$[0.6, 2.4]$
reduction factor of the velocity weights	$\tilde{\alpha}$	0.5

bounds for initializing and reinitializing positions of the velocity weights at all times. MSOPSO is configured according to Table 3.6.

Table 3.6: MSOPSO Settings

Parameter	Label	Setting
absolute lower inertia weight bound	\check{w}_l	0.3
absolute upper inertia weight bound	\check{w}_u	0.9
inertia weight bound width	\check{w}_w	0.2
inertia weight bound flag	\check{w}_f	-1
absolute lower personal best weight bound	\check{c}_{1l}	0.5
absolute upper personal best weight bound	\check{c}_{1u}	2.5
personal best weight bound width	\check{c}_{1w}	0.2
personal best weight bound flag	\check{c}_{1f}	-1
absolute lower global best weight bound	\check{c}_{2l}	0.6
absolute upper global best weight bound	\check{c}_{2u}	2.4
global best weight bound width	\check{c}_{2w}	0.2
global best weight bound flag	\check{c}_{2f}	1

3.2.7 Repulsive Step-Optimized Particle Swarm Optimization

The implementation of RSOPSO follows the implementation of SOPSO. RSOPSO differs from SOPSO by using two phases. The idea of RSOPSO is to move the velocity weights towards convergence-yielding velocity weights in the attractive phase and towards diversity-yielding velocity weights in the repulsive phase. Phases are changed based on the mean separation. Further, RSOPSO uses Equation (3.9) as the objective function for the velocity weights instead of Equation (3.2) used by SOPSO. RSOPSO provides the option to stop the algorithm after a set number of complete phase cycles, i.e., completion of one attractive phase and one repulsive phase, without improvement in the global best position and value. Because we compare the PSO variants by looking at the results given a fixed number of function evaluations, RSOPSO is stopped if the iteration number n reaches the total number of iterations. RSOPSO is configured according to Table 3.7.

3.2.8 Moving Bound Repulsive Step-Optimized PSO

The implementation of MRSOPSO follows the implementation of SOPSO, RSOPSO, and MSOPSO closely. The search space used for optimizing the velocity weights is calculated at every iteration

Table 3.7: RSOPSO Settings

Parameter	Label	Setting
mean separation absolute upper threshold	$s_u(n)$	mean separation after initialization
mean separation absolute lower threshold	$s_l(n)$	$\frac{s_u(n)}{100}$
mean separation absolute lower divisor	\check{s}_l	10
mean separation absolute upper divisor	\check{s}_u	2.5

and used to determine the bounds for initializing and reinitializing position of the velocity weights. MRSOPSO is configured according to Table 3.8.

3.3 Software Package

From the many known PSO variants, we implemented GBPSO [28], DWPSO [13], TVACPSO [55], and GCP SO [64] and use them for comparison. We further implemented the proposed APSO variants SOPSO, MSOPSO, RSOPSO, and MRSOPSO. The PSO variants implemented can be used in different combinations; see Appendix A. The arguments of the PSO software package implemented in Fortran are described in Appendix A, and a sample call to the PSO software package is found in Appendix B.

Although we only test the implemented PSO software package on common unimodal and multimodal continuous optimization test problems, the implemented PSO software package can be used to solve convex, linear programming, smooth non-linear, quadratic programming, mixed-integer, combinatorial, global, non-smooth, dynamic, and multi-objective optimization problems.

All PSO variants in the implemented PSO software package can be used to solve global and non-smooth optimization problems [23, 73]. The proposed RSOPSO can be used to solve dynamic optimization problems because it uses a repulsive phase. Although the implemented software package can be applied to solve smooth non-linear optimization and quadratic programming problems, there are optimization techniques specialized for solving smooth non-linear and quadratic programming optimization problems. Such optimization techniques usually use gradient information. The implemented PSO software package can be used for solving convex and linear programming optimization problems. However, there are specialized techniques that can be expected to solve convex optimization problems more efficiently using the specific structure of such problems.

The implemented PSO software package does not implement any PSO variant that can find the Pareto front to multi-objective problems, but it could be extended to include such a PSO variant.

The implemented PSO software package cannot be used to solve dynamic dimension optimization problems, but it could be extended to address such problems.

Table 3.8: MRSOPSO Settings

Parameter	Label	Setting
absolute lower inertia weight bound for attractive phase	\check{w}_l	0.3
absolute upper inertia weight bound for attractive phase	\check{w}_u	0.9
inertia weight bound width for attractive phase	\check{w}_w	0.2
inertia weight bound flag for attractive phase	\check{w}_f	-1
absolute lower personal best weight bound for attractive phase	\check{c}_{1l}	0.5
absolute upper personal best weight bound for attractive phase	\check{c}_{1u}	2.5
personal best weight bound width for attractive phase	\check{c}_{1w}	0.2
personal best weight bound flag for attractive phase	\check{c}_{1f}	-1
absolute lower global best weight bound for attractive phase	\check{c}_{2l}	0.6
absolute upper global best weight bound for attractive phase	\check{c}_{2u}	2.4
global best weight bound width for attractive phase	\check{c}_{2w}	0.2
global best weight bound flag for attractive phase	\check{c}_{2f}	1
absolute lower inertia weight bound for repulsive phase	\check{w}_l	0.3
absolute upper inertia weight bound for repulsive phase	\check{w}_u	0.9
inertia weight bound width for repulsive phase	\check{w}_w	0.2
inertia weight bound flag for repulsive phase	\check{w}_f	-1
absolute lower personal best weight bound for repulsive phase	\check{c}_{1l}	-2.5
absolute upper personal best weight bound for repulsive phase	\check{c}_{1u}	-0.5
personal best weight bound width for repulsive phase	\check{c}_{1w}	0.2
personal best weight bound flag for repulsive phase	\check{c}_{1f}	1
absolute lower global best weight bound for repulsive phase	\check{c}_{2l}	-2.5
absolute upper global best weight bound for repulsive phase	\check{c}_{1u}	-0.5
global best weight bound width for repulsive phase	\check{c}_{2w}	0.2
global best weight bound flag for repulsive phase	\check{c}_{2f}	-1

CHAPTER 4

EXPERIMENTS

To compare the implemented PSO variants, 22 optimization benchmark problems have been selected based on their use in the literature. PSO variants that perform well for the 22 optimization benchmark problems will likely perform well for other optimization problems such as found in industry because the benchmark problems cover a wide range of topographies. This chapter includes descriptions of the 22 optimization benchmark problems in Section 4.1, the experimental setup in Section 4.2, and the optimization results in Section 4.3.

4.1 Test Functions

Twenty-two optimization test problems have been implemented and are used to compare different PSO and proposed APSO variants. The sources for the benchmark problems are as follows. All the test problems from the semi-continuous challenge [8] have been implemented, including the Ackley test problem Equation (4.1), the Alpine test problem Equation (4.2), the Griewank test problem Equation (4.9), the Parabola test problem Equation (4.14), the Rosenbrock test problem Equation (4.16), and the Tripod test problem Equation (4.22).

Further, some of the optimization test problems described in [41] have been selected based on their topography to guarantee a diverse set of problems, including the Six-hump Camel Back test problem Equation (4.3), the De Jong 5 test problem Equation (4.4), the Deceptive test problem Equation (4.5), the Drop Wave test problem Equation (4.6), the Easom test problem Equation (4.7), the Goldstein–Price test problem Equation (4.10), the Axis Parallel Hyper-ellipsoid test problem Equation (4.11), the Michalewicz test problem Equation (4.12), and the Shubert test problem Equation (4.19) [8].

We also use some of the optimization test problems from [81], where we selected diverse test problems to expand our benchmark set. These include the Generalized Penalized test problem Equation (4.8), the Non-continuous Rastrigin test problem Equation (4.13), the Rastrigin test problem Equation (4.15), the Schwefel’s P2.22 test problem Equation (4.18), the Sphere test problem Equation (4.20), and the Step test problem Equation (4.21) [81].

We use one optimization problem from [40], namely Schaffer’s F6 test problem Equation (4.17).

In the following equations, $\mathbf{x} \in \mathbb{R}^d$ is the position of a particle, x_d is the component of \mathbf{x} at index d , and auxiliary functions are labeled g . For uniformity, benchmark problems that have a optimum different from 0 were shifted so that their optimum is located at 0.

The Ackley test problem has the following objective function [8]:

$$f_1(\mathbf{x}) = -20 \exp \left(-0.2 \sqrt{\frac{\sum_{d=1}^D x_d^2}{D}} \right) - \exp \left(\frac{\sum_{d=1}^D \cos(2\pi x_d)}{D} \right) + 20 + \exp(1). \quad (4.1)$$

Ackley is used in the search space $[-30, 30]^{30}$ and its minimum is given in [8, 81]. Ackley can be generalized to search spaces of arbitrary dimension. Ackley is said to be difficult to solve because the global optimum has a narrow neighborhood [8]. Figure 4.1 shows Ackley for two dimensions.

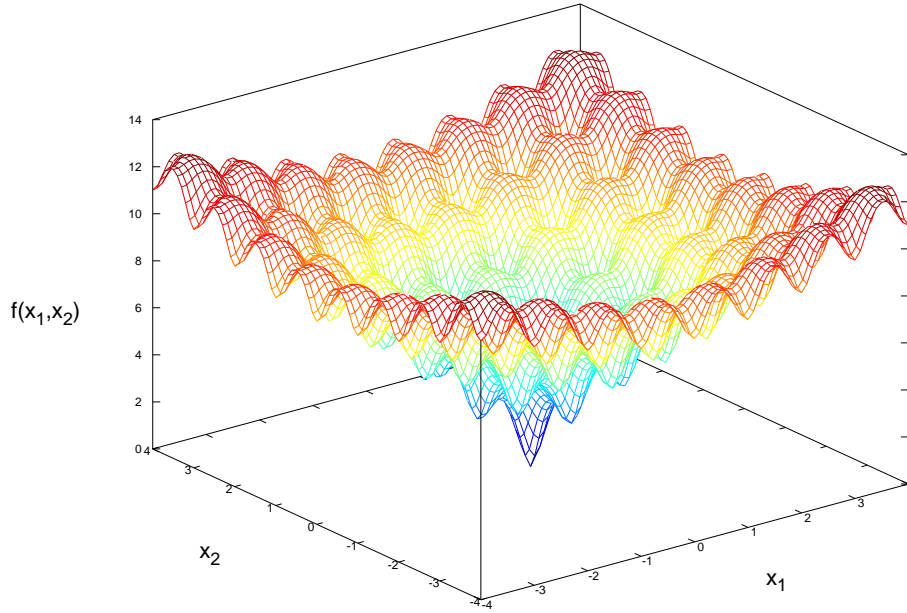


Figure 4.1: The topography of Ackley, Equation (4.1), for two dimensions.

The Alpine test problem has the following objective function [8]:

$$f_2(\mathbf{x}) = \sum_{d=1}^D |x_d \sin(x_d) + 0.1x_d|. \quad (4.2)$$

Alpine is used in the search space $[-10, 10]^{10}$ and its minimum is given in [8]. Alpine can be generalized to search spaces of arbitrary dimension. Alpine has multiple local optima [8]. Figure 4.2 shows Alpine for two dimensions.

The Six-hump Camel Back test problem has the following objective function [41]:

$$f_3(x_1, x_2) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3} \right) x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2 + 1.03162845348987734475. \quad (4.3)$$

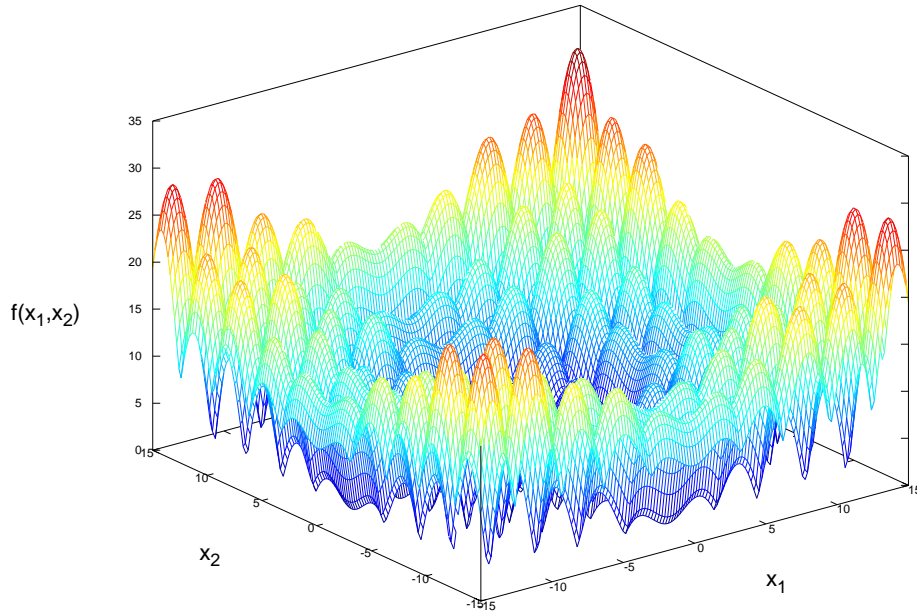


Figure 4.2: The topography of Alpine, Equation (4.2), for two dimensions.

Six-hump camel back is used in the search space $[-2, 2]^2$ and the optimum is given in [41]. Six-hump Camel Back cannot be generalized to search spaces of arbitrary dimension. Two of the six local minima of Six-hump Camel Back are global minima [41].

The De Jong 5 test problem has the following objective function [41]:

$$f_4(x_1, x_2) = \left(0.002 + \sum_{d=1}^{25} [\check{d} + (x_1 - \check{a}_d)^6 + (x_2 - \check{b}_d)^6]^{-1} \right)^{-1} - 0.99800383779444934440, \quad (4.4)$$

where the constant vectors $\check{\mathbf{a}}$ and $\check{\mathbf{b}}$ are set according to Appendix C. De Jong 5 is used in the search space $[-65.536, 65.536]^2$ and the optimum is given in [11]. De Jong 5 cannot be generalized to search spaces of arbitrary dimension. De Jong 5 is multimodal [41].

The Deceptive Type 3 test problem has the following objective function [41]:

$$f_5(\mathbf{x}) = - \left[\frac{1}{n} \sum_{d=1}^{30} g_1(x_d) \right]^\beta, \quad (4.5)$$

$$\text{where } g_1(x_d) = \begin{cases} -\frac{x_d}{\check{c}_d} + \frac{4}{5}, & \text{if } 0 \leq x_d \leq \frac{4}{5}\check{c}_d, \\ \frac{5x_d}{\check{c}_d} - 4, & \text{if } \frac{4}{5}\check{c}_d < x_d \leq \check{c}_d, \\ \frac{5*(x_d - \check{c}_d)}{\check{c}_d - 1} + 1, & \text{if } \check{c}_d < x_d \leq \frac{1+4\check{c}_d}{5}, \\ \frac{x_d - 1}{1 - \check{c}_d} + \frac{4}{5}, & \text{if } \frac{1+4\check{c}_d}{5} < x_d \leq 1, \end{cases}$$

where $\beta = 2.5$ and $\check{\mathbf{c}}$ is a constant vector containing unique numbers randomly selected and set according to Appendix C. Setting $\check{\mathbf{c}}$ guarantees reproducible and comparable results for solving Deceptive Type 3 with different solvers. Deceptive Type 3 is used in the search space $[0, 1]^{30}$ and its minimum is determined via optimization using multiple PSO variants and many number of function evaluations. Deceptive Type 3 cannot be generalized to search spaces of arbitrary dimension. Deceptive Type 3 has a total of $3^{30} - 1$ local optima and the neighborhoods of the local optima are $5^{30} - 1$ times larger than the neighborhood of the global optimum [41].

The Drop Wave test problem has the following objective function [41]:

$$f_6(x_1, x_2) = -\frac{1 + \cos(12 + \sqrt{x_1^2 + x_2^2})}{\frac{1}{2}(x_1^2 + x_2^2) + 2} + 1. \quad (4.6)$$

Drop Wave is used in the search space $[-5.12, 5.12]^2$ and the optimum is given in [57]. Drop Wave cannot be generalized to search spaces of arbitrary dimension. Drop Wave is multimodal [41].

The Easom test problem has the following objective function [41]:

$$f_7(x_1, x_2) = -\cos(x_1) \cos(x_2) \exp^{-(x_1 - \pi)^2 - (x_2 - \pi)^2} + 1. \quad (4.7)$$

Easom is used in the search space $[-100, 100]^2$ and the optimum is given in [41]. Easom cannot be generalized to search spaces of arbitrary dimension. Easom has only one minimum, but the area leading towards this minimum is small compared to the complete search space [41].

The Generalized Penalized test problem has the following objective function [81]:

$$\begin{aligned}
f_8(\mathbf{x}) = & \frac{\pi}{D} \left[10 \sin^2(\pi g_1(x_1)) \right. \\
& + \sum_{d=1}^{D-1} [(g_1(x_d) - 1)^2 (1 + 10 \sin^2(\pi g_1(x_{d+1}))) \\
& \left. + (g_1(x_d) - 1)^2 \right] \\
& + \sum_{d=1}^D g_2(x_d, 10, 100, 4), \tag{4.8}
\end{aligned}$$

$$\text{where } g_1(z) = 1 + \frac{1}{4}(z + 1),$$

$$g_2(y_1, y_2, y_3, \delta) = \begin{cases} y_3(y_1 - y_2)^\delta, & \text{if } y_1 > y_2, \\ 0, & \text{if } -y_2 \leq y_1 \leq y_2, \\ y_3(-y_1 - y_2)^\delta, & \text{if } y_1 < -y_2, \end{cases}$$

where the auxiliary function g_2 has exactly one input value labeled z and the auxiliary function g_3 has exactly four input values labeled $y_1, y_2, y_3,$ and δ . Generalized Penalized is used in the search space $[-50, 50]^{30}$ and its minimum is given in [81]. Generalized Penalized can be generalized to search spaces of arbitrary dimension. Generalized Penalized is multimodal [81].

The Griewank test problem has the following objective function [8]:

$$f_9(\mathbf{x}) = \frac{\sum_{d=1}^D (x_d - 100)^2}{4000} - \prod_{d=1}^D \cos\left(\frac{x_d - 100}{\sqrt{d}}\right) + 1. \tag{4.9}$$

Griewank is used in the search space $[-300, 300]^{30}$ and its minimum is given in [8, 81]. Griewank can be generalized to search spaces of arbitrary dimension. Griewank has many local optima with small neighborhoods [8].

The Goldstein–Price test problem has the following objective function [41]:

$$\begin{aligned}
f_{10}(x_1, x_2) = & \left[1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2) \right] \\
& \left[30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2) \right] - 3. \tag{4.10}
\end{aligned}$$

Goldstein–Price is used in the search space $[-2, 2]^2$ and the optimum is given in [41]. Goldstein–Price cannot be generalized to search spaces of arbitrary dimension. Goldstein–Price is unimodal [41].

The Axis parallel Hyper-ellipsoid test problem has the following objective function [41]:

$$f_{11}(\mathbf{x}) = \sum_{d=1}^D (dx_d^2). \tag{4.11}$$

Axis parallel hyper-ellipsoid is used in the search space $[-5.12, 5.12]^{100}$ and its minimum is given in [51]. Axis parallel Hyper-ellipsoid can be generalized to search spaces of arbitrary dimension. Axis parallel Hyper-ellipsoid is continuous, convex, and unimodal [41].

The Michalewicz test problem has the following objective function [41]:

$$f_{12}(\mathbf{x}) = -\sum_{d=1}^D \sin(x_d) \left[\sin\left(\frac{dx_d^2}{\pi}\right) \right]^{2m} + 9.6601517156413477494, \quad (4.12)$$

where m defines the steepness of slopes, and larger m settings yield increased steepness and difficulty [41]. Michalewicz is used in the search space $[0, \pi]^{10}$, m is set to 10, and the optimum is given in [41]. Michalewicz can be generalized to search spaces of arbitrary dimension. Michalewicz has $D!$ local optima [41].

The Non-continuous Rastrigin test problem has the following objective function [81]:

$$f_{13}(\mathbf{x}) = \sum_{d=1}^D [g_1(x_d)^2 - 10 \cos(2\pi g_1(x_d)) + 10], \quad (4.13)$$

$$\text{where } g_1(z) = \begin{cases} z, & \text{if } |z| < 0.5, \\ \frac{\text{round}(2z)}{2}, & \text{if } |z| \geq 0.5, \end{cases}$$

and round is a function rounding the given value to the next closest integer value. Non-continuous Rastrigin is used in the search space $[-5.12, 5.12]^{30}$ and its minimum is given in [81]. Non-continuous Rastrigin can be generalized to search spaces of arbitrary dimension. Non-continuous Rastrigin is multimodal [81].

The Parabola test problem has the following objective function [8]:

$$f_{14}(\mathbf{x}) = \sum_{d=1}^D x_d^2. \quad (4.14)$$

Parabola is used in the search space $[-20, 20]^{200}$ and its minimum is given in [8]. Parabola can be generalized to search spaces of arbitrary dimension. Parabola is a convex optimization problem and should be solvable by local optimization techniques. However, random-based optimization techniques may not be able to solve the problem accurately [8]. Figure 4.3 shows Parabola for two dimensions.

The Rastrigin test problem has the following objective function [55]:

$$f_{15}(\mathbf{x}) = \sum_{d=1}^D [x_d^2 - 10 \cos(2\pi x_d) + 10]. \quad (4.15)$$

Rastrigin is used in the search space $[-10, 10]^{30}$ and its minimum is given in [41, 81]. Rastrigin can be generalized to search spaces of arbitrary dimension. Rastrigin is based on De Jong 5, but it has more local minima due to the cosine term [41, 55]. Figure 4.4 shows Rastrigin for two dimensions.

The Rosenbrock test problem has the following objective function [8]:

$$f_{16}(\mathbf{x}) = \sum_{d=1}^{D-1} \left((1 - x_d)^2 + 100(x_d^2 - x_{d+1})^2 \right). \quad (4.16)$$

Rosenbrock is used in the search space $[-10, 10]^{30}$ and its minimum is given in [8]. Rosenbrock can be generalized to search spaces of arbitrary dimension. Rosenbrock is said to be misleadingly flat

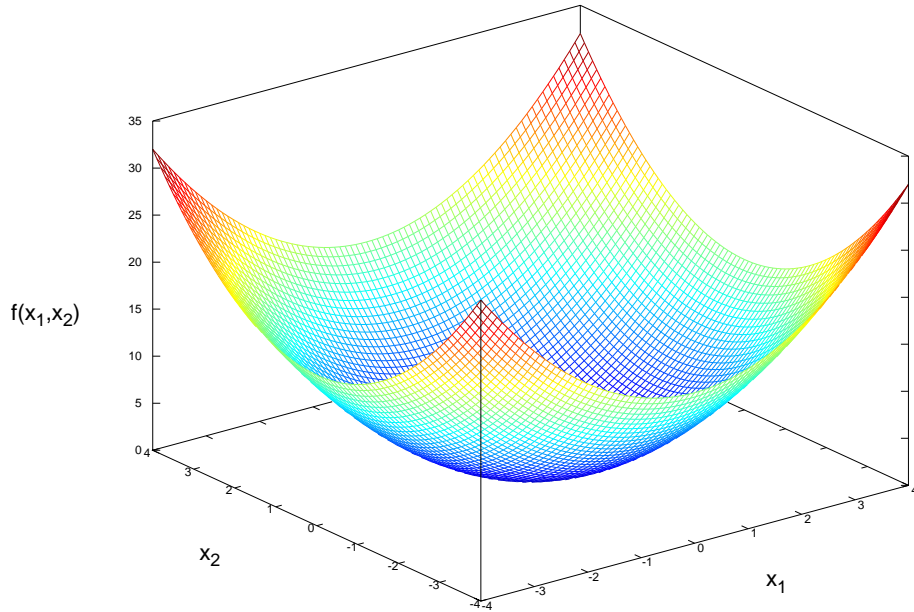


Figure 4.3: The topography of Parabola, Equation (4.14), for two dimensions.

towards the global optimum, and the exact optimum is hard to find for high dimensions [8]. Figure 4.5 shows Rosenbrock for two dimensions.

The Schaffer's F6 test problem has the following objective function [40]:

$$f_{17}(x_1, x_2) = 0.5 + \frac{\sin^2(\sqrt{x_1^2 + x_2^2}) - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2}. \quad (4.17)$$

Schaffer's F6 is used in the search space $[-100, 100]^2$ and its minimum is given in [39]. Schaffer's F6 cannot be generalized to search spaces of arbitrary dimension. Schaffer's F6 is multimodal [33, 40].

The Schwefel's P2.22 test problem has the following objective function [81]:

$$f_{18}(\mathbf{x}) = \sum_{d=1}^D |x_d| + \prod_{d=1}^D |x_d|. \quad (4.18)$$

Schwefel's P2.22 is used in the search space $[-10, 10]^{30}$ and its minimum is given in [81]. Schwefel's P2.22 can be generalized to search spaces of arbitrary dimension. Schwefel's P2.22 is unimodal [81].

The Shubert test problem has the following objective function [41]:

$$f_{19}(x_1, x_2) = -\sum_{d=1}^5 d \cos((d+1)x_1 + d) \sum_{d=1}^5 d \cos((d+1)x_2 + d) + 186.73090883102399029667. \quad (4.19)$$

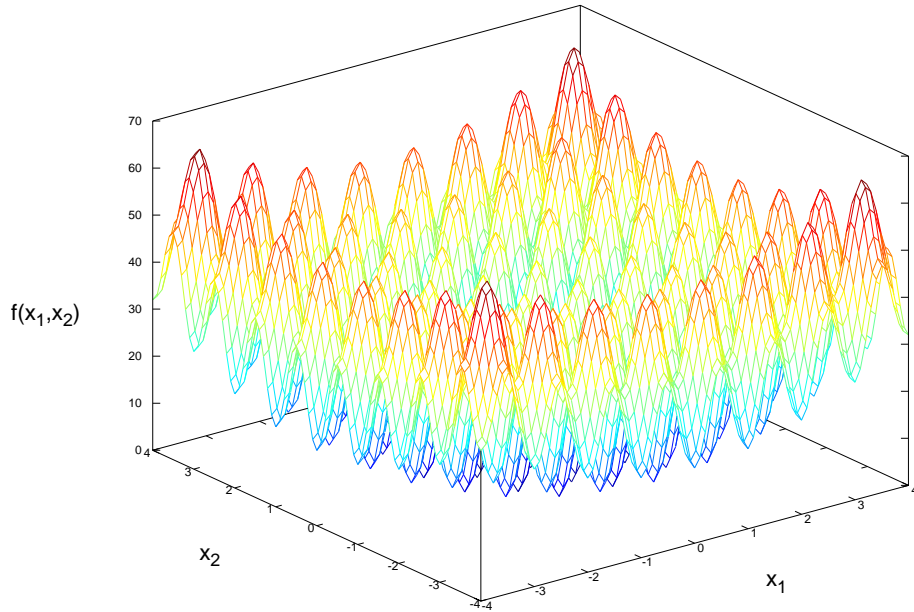


Figure 4.4: The topography of Rastrigin, Equation (4.15), for two dimensions.

Shubert is used in the search space $[-10, 10]^2$ and the optimum is given in [19]. Shubert cannot be generalized to search spaces of arbitrary dimension. Shubert is multimodal [41].

The Sphere test problem has the following objective function [55]:

$$f_{20}(\mathbf{x}) = \sum_{d=1}^D x_d^2. \quad (4.20)$$

Sphere is used in the search space $[-100, 100]^{100}$ and its minimum is given in [81]. Sphere can be generalized to search spaces of arbitrary dimension. Sphere is unimodal [55].

The Step test problem has the following objective function [81]:

$$f_{21}(\mathbf{x}) = \sum_{d=1}^D ([x_d + 0.5])^2, \quad (4.21)$$

where $\lfloor \cdot \rfloor$ is the floor function, meaning $\lfloor x_d + 0.5 \rfloor$ results in the largest integer y such that $y \leq x_d + 0.5$. Step is used in the search space $[-100, 100]^{30}$ and its minimum is given in [81]. Step can be generalized to search spaces of arbitrary dimension. Step is unimodal [81].

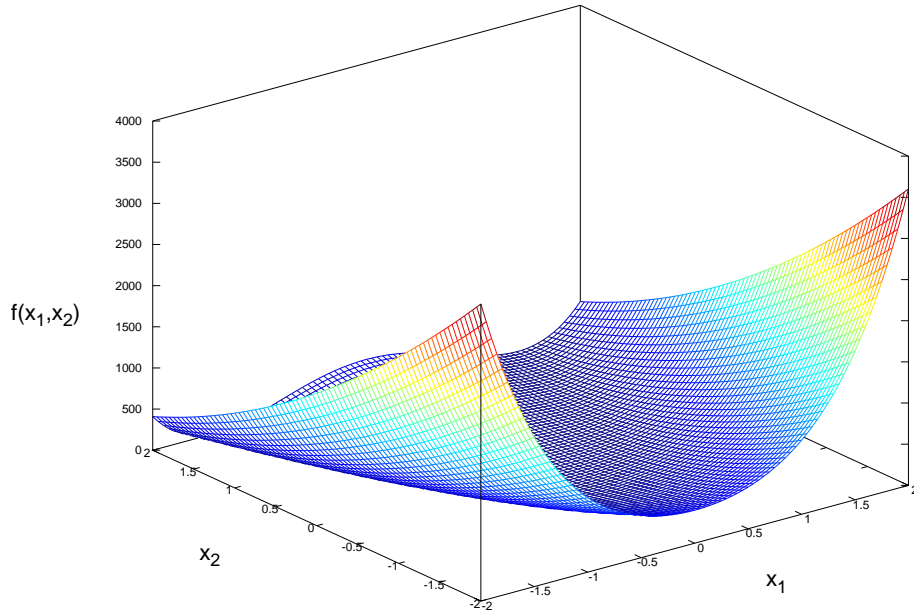


Figure 4.5: The topography of Rosenbrock, Equation (4.16), for two dimensions.

The Tripod test problem has the following objective function [8]:

$$\begin{aligned}
 f_{22}(x_1, x_2) &= g_1(x_2)(1 + g_1(x_1)) + |x_1 + 50g_1(x_2)(1 - 2g_1(x_1))| \\
 &\quad + |x_2 + 50(1 - 2g_1(x_2))|, \\
 \text{where } g_1(z) &= \begin{cases} 1, & \text{if } z \geq 0, \\ 0, & \text{if } z < 0. \end{cases}
 \end{aligned} \tag{4.22}$$

Tripod is used in the search space $[-100, 100]^2$ and its minimum is given in [8]. Tripod cannot be generalized to search spaces of arbitrary dimension. Tripod is discontinuous [8]. Even though Tripod has only two dimensions, global optimization techniques might get trapped in the large neighborhoods of the two local minima [8].

4.2 Experimental Setup

If not stated otherwise, the parameters are set to the values described in Section 3.2. We compare all PSO variants on four test cases using four different, fixed numbers of function evaluations. The first test uses $n_p = 100$ particles and $N = 74,999$ iterations for a total of 7,500,000 function evaluations (including initialization). The second test uses $n_p = 100$ particles and $N = 14,999$

iterations N for a total of 1,500,000 function evaluations (including initialization). The third test uses $n_p = 100$ particles and $N = 2,499$ iterations for a total of 250,000 function evaluations (including initialization). The fourth test uses $n_p = 30$ particles and $N = 99$ iterations for a total of 3,000 function evaluations (including initialization). We use double precision instead of real precision. Double precision has 15 significant digits, a sign indicator, and a range of approximately 10^{-308} to 10^{308} , compared to real which has 7 significant digits, a sign indicator, and a range of approximately 10^{38} to 10^{-38} [70]. The accuracy has an influence on the optimization results as shown by rudimentary experiments. The reason for choosing double precision is that the results are less affected by rounding errors, thus increasing the versatility of the PSO software package. Due to space considerations, solutions are reported to an accuracy of 10 decimal places.

4.3 Results

Optimization techniques are usually evaluated by comparing them to other optimization techniques in terms of the number of function evaluations required to find a given optimum or the best objective value found [69]. We follow the approach of reporting the optimum given a certain number of function evaluations. We particularly focus on results given for large numbers of function evaluations (FE), i.e., one to several million, because the proposed APSO variants were developed with a focus on efficiently and effectively solving difficult problems that require such high numbers of function evaluations for their solution.

We look at the minimum and average solutions. The minimum solutions are the best solutions from three optimization runs using different random seeds. The average (mean) solutions are the average over the solutions from three optimization runs using different random seeds. The minimum solutions are important if users are interested in the real optimum. The average solutions are important for applications where the objective function is not certain or only an approximation to the real world, such as for minimizing the risk of a portfolio. In that case, the average might be a more useful measure for the user. Additionally, the average can give insights into the robustness of the optimization technique; i.e., independence of the solution from the random seeds used.

Tables 4.1–4.4 compare the minimum solutions found by the implemented known PSO and proposed APSO variants by means of wins, draws, and losses. A particular PSO or APSO variant gets a win in case it only found the best solution, a draw in case it and at least another PSO or APSO variant found the best solution, and a loss in case the PSO or APSO variant did not find the best solution. Similarly, Tables D.1–D.4 in the Appendix D compare the average solutions by means of wins, draws, and losses. Because the tables that compare the different techniques by count of wins, draws, and losses compare as many as eight different solvers at the same time, and wins, draws, and losses do not consider the relative difference that yielded them, these tables should only

Table 4.1: Count of wins, draws, and losses for 7,500,000 FE considering minimum solutions.

Solver	win	draw	loss
GBPSO	0	18	4
DWPSO	0	19	3
TVACPSO	0	19	3
GCPSO	0	18	4
SOPSO	0	22	0
MSOPSO	0	21	1
RSOPSO	0	22	0
MRSOPSO	0	19	3

Table 4.2: Count of wins, draws, and losses for 1,500,000 FE considering minimum solutions.

Solver	win	draw	loss
GBPSO	1	17	4
DWPSO	0	17	5
TVACPSO	0	17	5
GCPSO	0	18	4
SOPSO	0	21	1
MSOPSO	0	18	4
RSOPSO	0	21	1
MRSOPSO	0	17	5

be interpreted using the other tables of this chapter; i.e., Tables 4.5–4.16 and D.5–D.16.

Tables 4.1 and D.1 compare the implemented known PSO and proposed APSO variants for 7,500,000 FE. The proposed SOPSO and RSOPSO win this comparison for large number of function evaluations by suffering no losses considering the minimum solutions as shown in Table 4.1. These results are supported by the comparison considering the average solutions, where SOPSO and RSOPSO win by suffering only 2 losses as shown in Table D.1. Based on these results and on the fact that SOPSO is faster than RSOPSO because it does not calculate the mean separation for every iteration, we recommend the use of SOPSO for problems that require large numbers of function evaluations such as several million. The timings for solving the optimization problems for 7,500,000 function evaluations showed that SOPSO is roughly 1.4 to 7.4 times faster than RSOPSO depending on the problem.

Tables 4.2 and D.2 compare the implemented known PSO and proposed APSO variants for 1,500,000 FE. SOPSO and RSOPSO win this comparison for high number of function evaluations by suffering only 1 loss considering the minimum solutions as shown in Table 4.2. Similarly, the result of the comparison considering the average solutions is won by the RSOPSO, which scores 1 win and only 3 losses as shown in Table D.2. SOPSO performs similarly well by suffering only 4 losses. Based on these results and the fact that SOPSO is faster than RSOPSO, we recommend the use of SOPSO for high numbers of function evaluations such as approximately one million.

Tables 4.3 and D.3 compare implemented known PSO and proposed APSO variants for 250,000 FE. GBPSO wins this comparison for low numbers of function evaluations by scoring 2 wins and only 5 losses considering the minimum solutions as shown in Table 4.3. SOPSO performs similarly well by scoring no wins and only 5 losses. This result is supported by the comparison considering the average solutions, where GBPSO wins by scoring 3 wins and only 7 losses as shown in Table

Table 4.3: Count of wins, draws, and losses for 250,000 FE considering minimum solutions.

Solver	win	draw	loss
GBPSO	2	15	5
DWPSO	0	16	6
TVACPSO	1	16	5
GCPSO	0	16	6
SOPSO	0	17	5
MSOPSO	1	15	6
RSOPSO	0	17	5
MRSOPSO	1	15	6

Table 4.4: Count of wins, draws, and losses for 3,000 FE considering minimum solutions.

Solver	win	draw	loss
GBPSO	1	5	16
DWPSO	0	4	18
TVACPSO	7	6	9
GCPSO	0	4	18
SOPSO	2	5	15
MSOPSO	4	3	15
RSOPSO	0	5	17
MRSOPSO	2	4	16

D.3. MSOPSO performs similarly by yielding 2 win and only 8 losses. Based on these results and the more detailed results of Tables 4.7, 4.11, 4.15, D.7, D.11, and D.15, we suggest users using the implemented PSO software package to either choose GBPSO or SOPSO if they require low numbers of function evaluations such as few hundred thousand.

Tables 4.4 and D.4 compare implemented known PSO and proposed APSO variants for 3,000 FE. TVACPSO wins this comparison for a few thousand function evaluations by scoring 7 wins and only 9 losses considering the minimum solutions as shown in Table 4.4. This result is supported by the comparison considering the average solutions, where TVACPSO wins by scoring 8 wins and only 9 losses as shown in Table D.4. Based on these results, we suggest users using the implemented PSO software package to choose TVACPSO if they require few thousand function evaluations.

Tables 4.5–4.8 compare the best solution found by GBPSO, DWPSO, TVACPSO, and GCPSO against the best solution found by SOPSO, RSOPSO, MSOPSO, and MRSOPSO given a certain number of function evaluations. If the known PSO variants found the same minimum to a given optimization test problem as the proposed APSO variants, that particular optimization test problem is not included in the table. Similarly, Tables D.5–D.8 in the Appendix D compare the best average solution found by any of the four implemented known PSO variants against the best average solution found by any of the four implemented proposed APSO variants. Again, if the known PSO variants found the same average optimum to a given optimization test problem as the proposed APSO variants, that particular optimization test problem is not included in the table.

Tables 4.5 and D.5 show promising results of the proposed APSO variants for 7,500,000 FE. The proposed APSO variants find the best objective values for two problems where the known PSO variants do not locate the optima. These results show that the proposed APSO variants clearly outperform the implemented known PSO variants for large number of function evaluations

considering the minimum solutions. These results are supported by the comparison considering the average solution, where the proposed APSO variants find the best objective value for four problems and the known PSO variants find the best objective value for one problem as shown in Table 4.5. Based on these results, we recommend the use of one proposed APSO for large numbers of function evaluations such as several million.

Table 4.5: PSO variants vs. APSO variants for 7,500,000 FE considering minimum solution.

Test Problem	known PSO variants	proposed APSO variants
Rastrigin	0.9949590571	0.0
Rosenbrock	0.0000114199	0.0

Tables 4.6 and D.6 show promising results of the proposed APSO variants for 1,500,000 FE. The proposed APSO variants find the best objective value for three problems compared to one for the known PSO variants. These results are supported by the comparison considering the average solution, where the proposed APSO variants find the best objective value for five problems compared to one for the known PSO variants. Based on these results, we recommend the use of one proposed APSO for high numbers of function evaluations such as approximately one million.

Table 4.6: PSO variants vs. APSO variants for 1,500,000 FE considering minimum solution.

Test Problem	known PSO variants	proposed APSO variants
Non-continuous Rastrigin	1.0	0.0
Parabola	0.0	0.0000215830
Rastrigin	8.9546315138	4.9747952855
Rosenbrock	0.0954049541	0.0000071796

Tables 4.7 and D.7 show a nearly equal performance of the known PSO variants and proposed APSO variants for 250,000 FE. The proposed APSO variants find the best objective value for three problems compared to three for the known PSO variants. These results are supported by the comparison considering the average solution, where the proposed APSO variants find the best objective value for five problems compared to four problems for the known PSO variants. Based on these results, we recommend the use of one of the proposed APSO or one of the known PSO for low numbers of function evaluations such as a few hundred thousand.

Tables 4.8 and D.8 show good performance of known PSO variants for 3,000 FE. The proposed APSO variants find the best objective value for eight problems compared to eight for the known PSO variants. Considering the average solution, the proposed APSO variants find the best objective

Table 4.7: PSO variants vs. APSO variants for 250,000 FE considering minimum solution.

Test Problem	known PSO variants	proposed APSO variants
Hyper-ellipsoid	0.0000001135	0.0000080629
Non-continuous Rastrigin	6.0	1.0
Parabola	9.3296371480	4.1970400021
Rastrigin	11.9395086851	10.9698748650
Rosenbrock	8.2346336506	18.6604566082
Sphere	0.0000000647	0.0001059541

value for six problems compared to eleven for the known PSO variants. Based on these results, we recommend the use of one known PSO for a few thousand function evaluations.

Tables 4.9–4.12 compare the best solutions found by SOPSO, MSOPSO, RSOPSO, and MRSOPSO; i.e., the tables compare the proposed APSO variants. If all four proposed APSO variants found the same minimum to a given optimization test problem, that particular optimization test problem is not included in the table. Similarly, Tables D.9–D.12 in the Appendix D compare the best average solutions found by SOPSO, MSOPSO, RSOPSO, and MRSOPSO. Again, if all four proposed APSO variants found the same average optimum to a given optimization test problem, that particular optimization test problem is not included in the table.

Tables 4.9 and D.9 compare APSO variants for 7,500,000 FE. Considering the minimum solution as shown in Table 4.9, SOPSO finds the best solution in three cases, MSOPSO finds the best solution in two cases, RSOPSO finds the best solution in three cases, and MRSOPSO finds the best solution in no cases. Considering the average solution as shown in Table D.9, SOPSO finds the best solution in four cases, MSOPSO finds the best solution in one case, RSOPSO finds the best solution in four cases, and MRSOPSO finds the best solution in one case. Based on these results and on that SOPSO is faster than RSOPSO, we recommend the use of SOPSO if one of the proposed APSO is used with several million function evaluations.

Tables 4.10 and D.10 compare the proposed APSO variants for 1,500,000 FE. Considering the minimum solution as shown in Table 4.10, SOPSO finds the best solution in five cases, MSOPSO finds the best solution in one case, RSOPSO finds the best solution in five cases, and MRSOPSO finds the best solution in no cases. Considering the average solution as shown in Table D.10, SOPSO finds the best solution in six cases, MSOPSO finds the best solution in one case, RSOPSO finds the best solution in seven cases, and MRSOPSO finds the best solution in four cases. Based on these results and on that SOPSO is faster than RSOPSO, we recommend the use of SOPSO if one of the proposed APSO is used with approximately one million function evaluations.

Table 4.8: PSO variants vs. APSO variants for 3,000 FE considering minimum solution.

Test Problem	known PSO variants	proposed APSO variants
Ackley	1.7535935539	2.5271868594
Alpine	0.0007749829	0.0003859068
Generalized Penalized	5.7588533203	8.4927368746
Griewank	1.0743647829	1.1644058859
Hyper-ellipsoid	1466.6333948600	1422.6001204600
Michalewicz	0.5441955682	0.3586014384
Non-continuous Rastrigin	72.5516666727	46.0000379643
Parabola	2530.0733023200	2480.6680864800
Rastrigin	84.1419055611	127.3033494730
Rosenbrock	262.9586718660	215.7469172450
Schaffer F6	0.0000001510	0.0003885796
Schwefel P2.22	2.7045438343	3.5953335808
Shubert	0.0000000006	0.0
Sphere	14235.7231372000	10954.0241478000
Step	48.0	73.0
Tripod	0.0000008724	0.0000152553

Table 4.9: Proposed APSO comparison for 7,500,000 FE considering minimum solutions.

Test Problem	SOPSO	MSOPSO	RSOPSO	MRSOPSO
Non-continuous Rastrigin	0.0	0.0	0.0	1.0
Rastrigin	0.0	0.0	0.0	3.9798362284
Rosenbrock	0.0	8.9247246703	0.0	15.0198240631

Table 4.10: Proposed APSO comparison for 1,500,000 FE considering minimum solutions.

Test Problem	SOPSO	MSOPSO	RSOPSO	MRSOPSO
Michalewicz	0.0	0.0	0.0	0.0398471887
Non-continuous Rastrigin	0.0	3.0	0.0	6.0
Parabola	0.0000215830	0.0141410461	0.0000215830	0.0008593517
Rastrigin	4.9747952855	10.9445496280	4.9747952855	7.9596724568
Rosenbrock	0.0000071796	0.7804030432	0.0000071796	4.7233727209

Tables 4.11 and D.11 compare the proposed APSO variants for 250,000 FE. Considering the minimum solution as shown in Table 4.11, SOPSO finds the best solution in four cases, MSOPSO finds the best solution in two cases, RSOPSO finds the best solution in four cases, and MRSOPSO finds the best solution in four cases. Considering the average solution as shown in Table D.11, SOPSO finds the best solution in six cases, MSOPSO finds the best solution in four cases, RSOPSO finds the best solution in six cases, and MRSOPSO finds the best solution in two cases. These results show that SOPSO and RSOPSO perform best for low numbers of function evaluations. Based on these results and on that SOPSO is faster than RSOPSO, we recommend the use of SOPSO if one of the proposed APSO is used with few hundred thousand function evaluations.

Table 4.11: Proposed APSO comparison for 250,000 FE considering minimum solutions.

Test Problem	SOPSO	MSOPSO	RSOPSO	MRSOPSO
Griewank	0.0	0.0073960421	0.0	0.0
Hyper-ellipsoid	0.0008089862	0.0259443600	0.0008089862	0.000080629
Michalewicz	0.0	0.0	0.0	0.1549133090
Non-continuous Rastrigin	1.0	8.0121094750	1.0	12.0
Parabola	41.8289099310	7.2349444553	6.6096926430	4.1970400021
Rastrigin	15.9193401243	10.9698748650	15.9193401243	13.9295140273
Rosenbrock	18.6604566082	23.7945552332	18.6604566082	22.3616219354
Sphere	0.0025875183	0.4123413985	0.0025875183	0.0001059541

Tables 4.12 and D.12 compare the proposed APSO variants for 3,000 FE. Considering the minimum solution as shown in Table 4.12, SOPSO finds the best solution in six cases, MSOPSO finds the best solution in eight cases, RSOPSO finds the best solution in four cases, and MRSOPSO finds the best solution in five cases. Considering the average solution as shown in Table D.12, SOPSO finds the best solution in six cases, MSOPSO finds the best solution in ten cases, RSOPSO finds the best solution in five cases, and MRSOPSO finds the best solution in seven cases. Based on these results, we recommend the use of MSOPSO if one of the proposed APSO is used with few thousand function evaluations.

Tables 4.13–4.16 compare the best solutions found by GBPSO, DWPSO, TVACPSO, and GCPSO; i.e., the tables compare the implemented known PSO variants. Tables D.13–D.16 in the Appendix D compare the average solutions found by GBPSO, DWPSO, TVACPSO, and GCPSO. Again, if all four known PSO variants found the same minimum or average solution to a given optimization test problem, that particular optimization test problem is not included in the table.

Tables 4.13 and D.13 compare PSO variants for 7,500,000 FE. Considering the minimum solution as shown in Table 4.13, GBPSO finds the best solution in two cases, DWPSO finds the best solution

Table 4.12: Proposed APSO comparison for 3,000 FE considering minimum solutions.

Test Problem	SOPSO	MSOPSO	RSOPSO	MRSOPSO
Ackley	4.1442049449	2.5271868594	4.1442049449	2.5749522063
Alpine	0.0003859068	0.0056322613	0.0022785970	0.0040930549
Drop Wave	0.0	0.0000000012	0.0	0.0000000101
Easom	0.0000000003	0.0000000001	0.0000000003	0.0
Gen... Penalized	11.9013451744	11.1778530240	11.9013451744	8.4927368746
Griewank	1.4105078803	1.1644058859	1.4105078803	1.1956777995
Goldstein-Price	0.0	0.0000000012	0.0	0.0000000001
Hyper-ellipsoid	2650.1932225500	1422.6001204600	2650.1932225500	1943.5890664600
Michalewicz	1.3477754937	0.4216800896	1.3477754937	0.3586014384
Non... Rastrigin	81.1015672036	46.0000379643	81.1015672036	87.3824192870
Parabola	3390.4652516200	2480.6680864800	3390.4652516200	2962.9491670500
Rastrigin	175.8517446580	127.3033494730	175.8517446580	135.6464539510
Rosenbrock	1029.2307493200	493.9609483910	1029.2307493200	215.7469172450
Schaffer F6	0.0003885796	0.0056596624	0.0003885796	0.0006701637
Schwefel P2.22	9.5389890792	3.5953335808	9.5389890792	7.1101007289
Shubert	0.0	0.0000000698	0.0000000001	0.0000000006
Sphere	16294.3398457000	10954.0241478000	16294.3398457000	15067.1131472000
Step	527.0	102.0	527.0	73.0
Tripod	0.0000152553	0.0004487209	0.0000152553	0.0000315142

in two cases, TVACPSO finds the best solution in two cases, and GCPSO finds the best solution in two cases. Considering the average solution as shown in Table D.13, GBPSO finds the best solution in one case, DWPSO finds the best solution in four cases, TVACPSO finds the best solution in three cases, and GCPSO finds the best solution in two cases. Based on these results, we recommend the use of DWPSO if one of the known PSO variants is used with several million function evaluations.

Table 4.13: PSO comparison for 7,500,000 FE considering minimum solutions.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO
Griewank	0.0	0.0073960403	0.0	0.0073960403
Michalewicz	0.2525725259	0.0	0.0	0.0
Non-continuous Rastrigin	6.0	0.0	3.0	0.0000000369
Rastrigin	25.8688951364	1.9899181142	3.9798362284	0.9949590571
Rosenbrock	0.0000114199	7.3443347199	13.9586009312	16.0244141123

Tables 4.14 and D.14 compare PSO variants for 1,500,000 FE. Considering the minimum solution as shown in Table 4.14, GBPSO finds the best solution in three cases, DWPSO finds the best solution in two cases, TVACPSO finds the best solution in one cases, and GCPSO finds the best solution in three cases. Considering the average solution as shown in Table D.14, GBPSO finds the best solution in four cases, DWPSO finds the best solution in four cases, TVACPSO finds the best solution in three cases, and GCPSO finds the best solution in four cases. Based on these results, we recommend the use of GBPSO or GCPSO if one of the known PSO variants is used with approximately one million function evaluations.

Table 4.14: PSO comparison for 1,500,000 FE considering minimum solutions.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO
Griewank	0.0	0.0	0.0073960403	0.0
Michalewicz	0.2525725259	0.0049111478	0.0	0.0
Non-continuous Rastrigin	6.0	4.0	11.0	1.0
Parabola	0.0	0.0000002112	0.0030505369	0.0000000870
Rastrigin	25.8688951364	8.9546315138	10.9445445902	9.9495855331
Rosenbrock	0.0954049541	19.6918183861	3.9869199112	4.1464319620

Tables 4.15 and D.15 compare PSO variants for 250,000 FE. Considering the minimum solution as shown in Table 4.15, GBPSO finds the best solution in four cases, DWPSO finds the best solution in one case, TVACPSO finds the best solution in three cases, and GCPSO finds the best solution in one case. Considering the average solution as shown in Table D.15, GBPSO finds the best solution

in five cases, DWPSO finds the best solution in three cases, TVACPSO finds the best solution in two cases, and GCPSO finds the best solution in four cases. Based on these results, we recommend the use of GBPSO if one of the known PSO variants is used with few hundred thousand function evaluations.

Table 4.15: PSO comparison for 250,000 FE considering minimum solutions.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO
Hyper-ellipsoid	0.0000001135	0.0061106210	0.0034030448	0.0024181041
Michalewicz	0.2574836737	0.0	0.0	0.0
Non-continuous Rastrigin	6.0	16.0000003717	22.0	21.0017979253
Parabola	9.3296371480	16.6223866133	17.7093667245	9.4273873954
Rastrigin	25.8688951364	15.9193398757	11.9395086851	24.8739663523
Rosenbrock	15.6856726242	18.4717862358	8.2346336506	22.6233668383
Sphere	0.0000000647	0.2551991869	0.1273730986	0.0402738382

Tables 4.16 and D.16 compare PSO variants for 3,000 FE. Considering the minimum solution as shown in Table 4.16, GBPSO finds the best solution in two cases, DWPSO finds the best solution in one case, TVACPSO finds the best solution in fifteen cases, and GCPSO finds the best solution in one case. Considering the average solution as shown in Table D.16, GBPSO finds the best solution in three cases, DWPSO finds the best solution in three cases, TVACPSO finds the best solution in sixteen cases, and GCPSO finds the best solution in three cases. Based on these results, we recommend the use of TVACPSO if one of the known PSO variants is used with few thousand function evaluations.

All figures compare the best of the proposed APSO variants against the best of the known PSO variants for selected, difficult optimization problems. The comparisons in Figures 4.6–4.8 consider minimum solutions and the comparisons in Figures 4.9–4.12 consider average solutions. All figures support the results of the previous sections; i.e., they show that the proposed APSO variants, in particular SOPSO, outperform the known PSO variants for increasing number of function evaluation. Because certain entities such as the inertia weight for DWPSO are different depending on the total number of iterations N used, results using more function evaluations may be worse (higher objective value) than results using fewer number of function evaluations.

Figure 4.6 shows the comparison of GCPSO and SOPSO depending on the number of function evaluations on Rastrigin considering the minimum solutions. Figure 4.7 shows the comparison of GBPSO and SOPSO on Rosenbrock considering the minimum solutions. Figure 4.8 shows the comparison of DWPSO and SOPSO depending on the number of function evaluations on Non-continuous Rastrigin considering the minimum solutions.

Table 4.16: PSO comparison for 3,000 FE considering minimum solutions.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO
Ackley	2.7864164356	2.7872954929	1.7535935539	3.4478779286
Alpine	0.0053181546	0.0285053980	0.0007749829	0.0243886490
Drop Wave	0.0	0.0000000008	0.0	0.0000000001
Easom	0.0000000957	0.0000000091	0.0	0.0000000184
Gen... Penalized	15.5799816557	23.2912583962	5.7588533203	19.6150447267
Griewank	1.3657400968	1.9707144262	1.0743647829	2.3992904992
Hyper-ellipsoid	1838.0238219300	2132.7192963100	1466.6333948600	1706.0779538200
Michalewicz	0.9988765596	0.5441955682	0.8847682876	0.9090513559
Non... Rastrigin	79.9054137431	96.2457432624	74.0262751250	72.5516666727
Parabola	3127.9040880900	3576.9163285500	2530.0733023200	2977.8936030400
Rastrigin	141.0597885160	182.3730264280	84.1419055611	123.8922789690
Rosenbrock	310.9464581860	1455.3539688200	262.9586718660	975.3149869380
Schaffer F6	0.0000001510	0.0097159308	0.0000003462	0.0097159099
Schwefel P2.22	4.1215510202	6.1703772446	2.7045438343	7.6256666346
Shubert	0.0000000242	0.0000000110	0.0000000006	0.0000000306
Sphere	16637.4774400000	16456.0655813000	14235.7231372000	21493.2244294000
Step	212.0	878.0	48.0	461.0
Tripod	0.0000762325	1.000070836	0.0000008724	0.0000162968

Figure 4.9 shows the comparison of TVACPSO and SOPSO depending on the number of function evaluations on Michalewicz considering the average solutions. Figure 4.10 shows the comparison of DWPSO and SOPSO depending on the number of function evaluations on Non-continuous Rastrigin considering the average solutions. Figure 4.11 shows the comparison of DWPSO and RSOPSO depending on the number of function evaluations on Rastrigin considering the average solutions. Figure 4.12 shows the comparison of GBPSO and SOPSO depending on the number of function evaluations on Rosenbrock considering the average solutions.

By roughly doubling the number of dimensions for all benchmark problems that allow arbitrary dimensionality, we looked at more difficult, higher-dimensional problems. In particular we used Ackley for 100 dimensions, Alpine for 30 dimensions, Generalized Penalized for 60 dimensions, Griewank for 60 dimensions, Hyper-ellipsoid for 200 dimensions, Michalewicz for 30 dimensions, Non-continuous Rastrigin for 60 dimensions, Parabola for 250 dimensions, Rastrigin for 60 dimensions, Rosenbrock for 60 dimensions, Schwefel P2.22 for 60 dimensions, Sphere for 200 dimensions, and Step for 60 dimensions. We optimized the more difficult problems using 15,000,000, 7,500,000, and 250,000 function evaluations. The results show that the proposed APSO variants outperform the known PSO variants for such problems, supporting the results of this thesis. The results can be found in Appendix E.

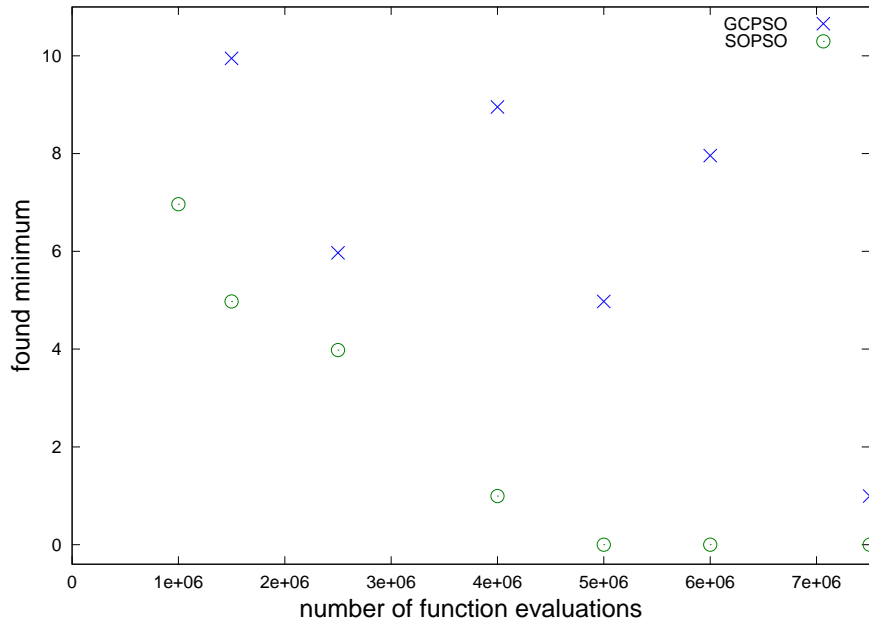


Figure 4.6: GCPSO vs. SOPSO on Rastrigin considering minimum solutions.

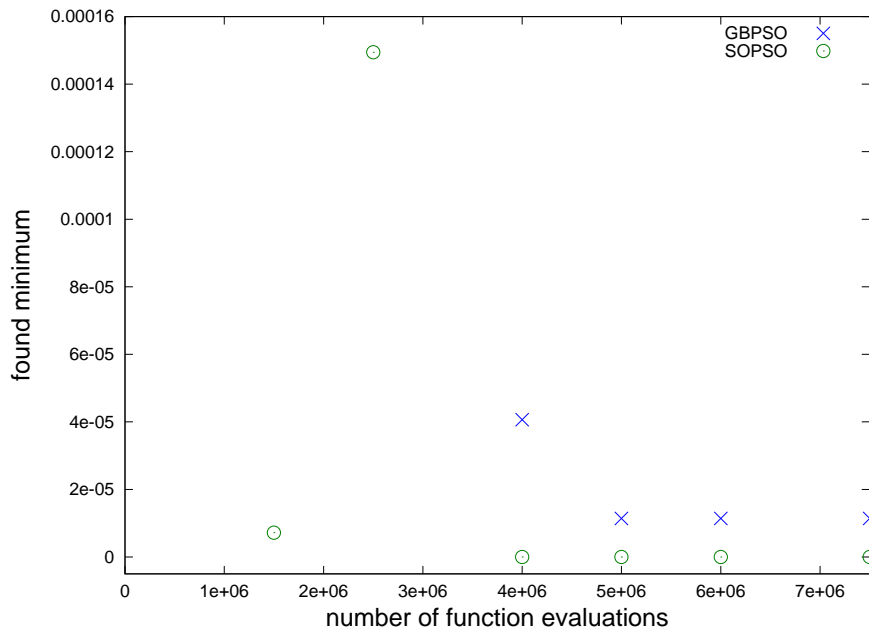


Figure 4.7: GBPSO vs. SOPSO on Rosenbrock considering minimum solutions.

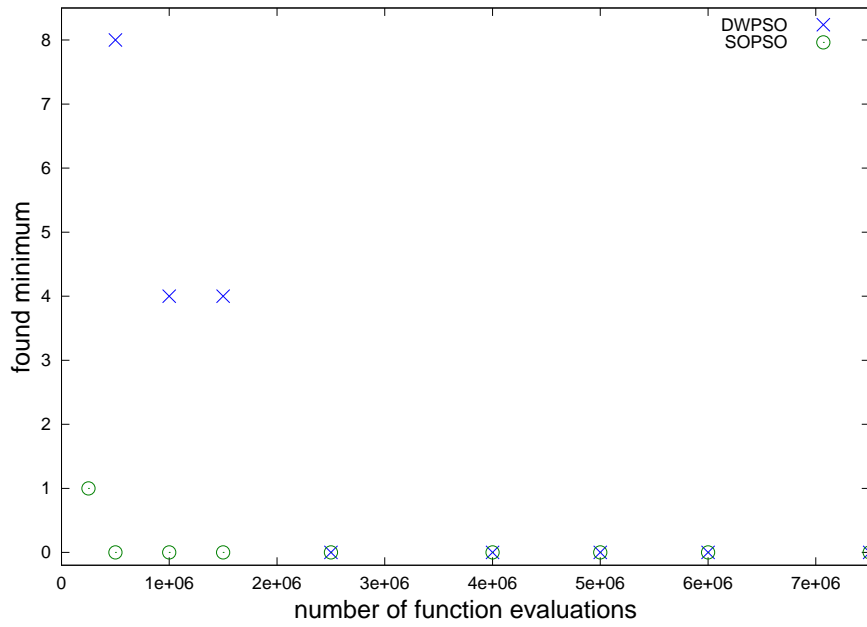


Figure 4.8: DWPSO vs. SOPSO on Non-continuous Rastrigin considering minimum solutions.

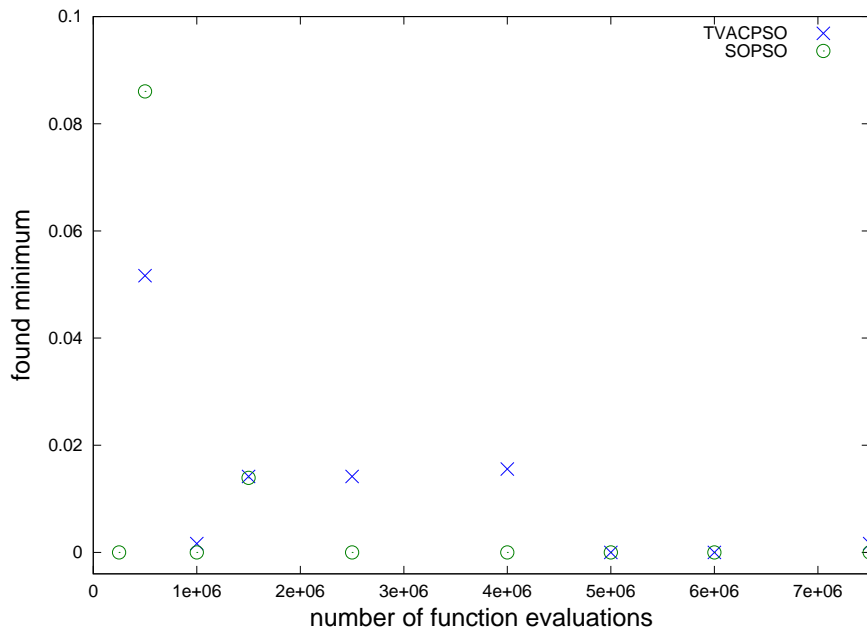


Figure 4.9: TVACPSO vs. SOPSO on Michalewicz considering average solutions.

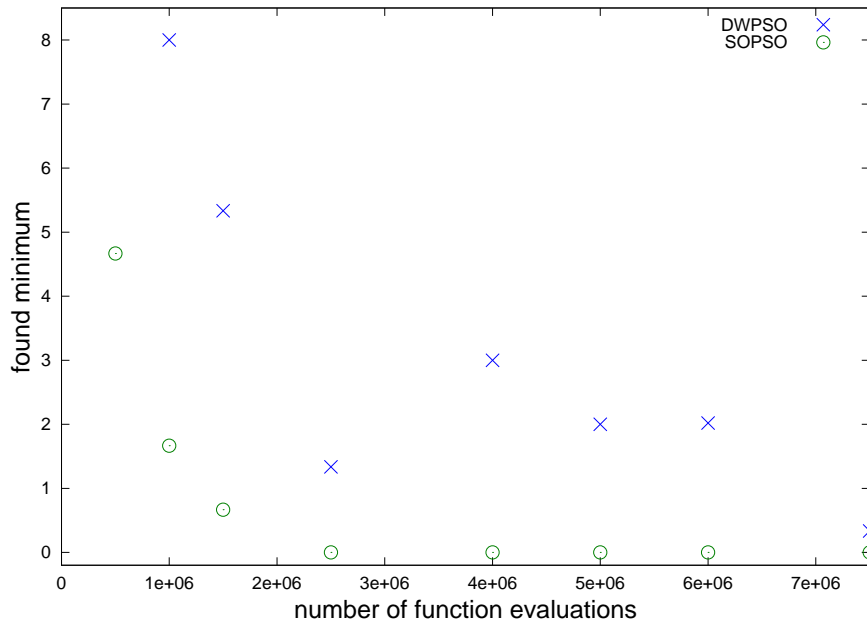


Figure 4.10: DWPSO vs. SOPSO on Non-continuous Rastrigin considering average solutions.

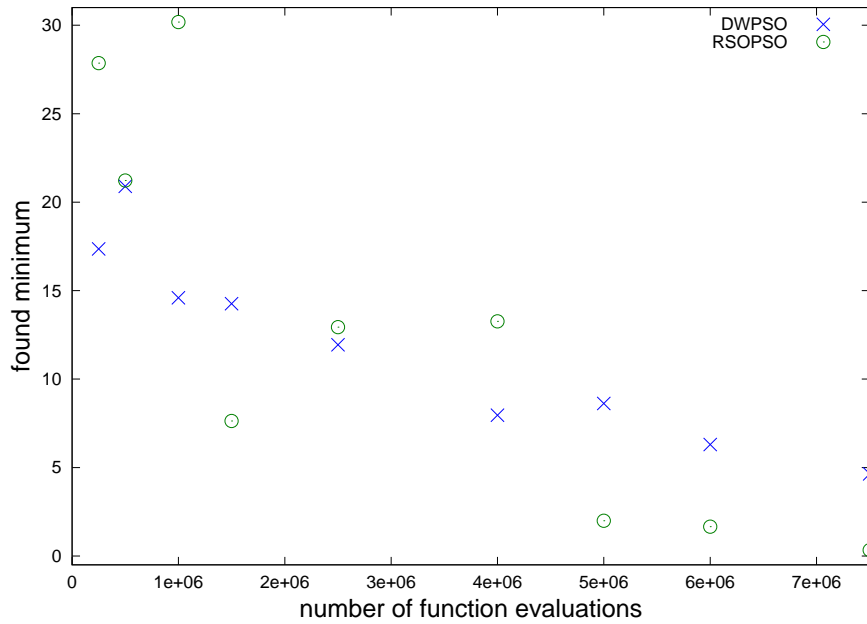


Figure 4.11: DWPSO vs. RSOPSO on Rastrigin considering average solutions.

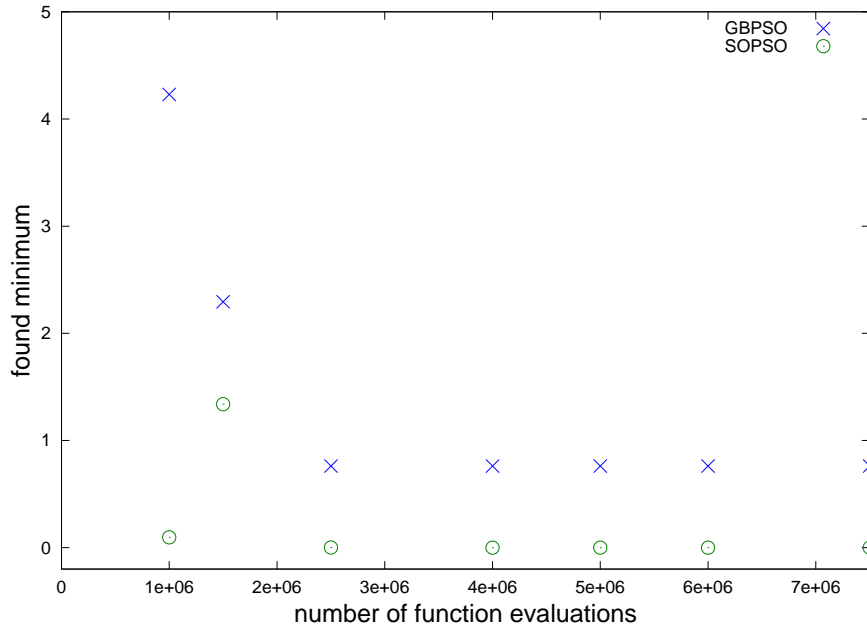


Figure 4.12: GBPSO vs. SOPSO on Rosenbrock considering average solutions.

4.3.1 Summary of Results

Given the results, we can conclude that the proposed APSO variants perform best for high numbers of function evaluations, on the order of several million, that are required for the solution of difficult optimization problems. For such difficult optimization problems the concept of optimizing the velocity weights improves the performance of PSO. The proposed APSO variants perform similarly to slightly better for lower numbers of function evaluations such as a few hundred thousand. For a few thousand function evaluations, the known PSO variants outperform the proposed APSO variants.

In particular, we recommend users to choose SOPSO if they would like to run the implemented PSO software package to solve hard problems that require high numbers of function evaluations such as on the order of one to several million. We recommend users to either choose GBPSO or SOPSO to solve problems that require low numbers of function evaluations such as a few hundred thousand. We recommend users to choose TVACPSO if they would like to run the implemented PSO software package to solve problems that require a few thousand function evaluations. For problems of unknown difficulty or no good estimate of the required number of function evaluations, we recommend to choose SOPSO and to use high numbers of function evaluations, such as one million or several million, and increase the numbers of function evaluations if necessary.

CHAPTER 5

CONCLUSION

We propose the four APSO variants SOPSO, MSOPSO, RSOPSO, and MRSOPSO. The idea behind SOPSO is to give every particle its own velocity weights and to treat the problem of finding good velocity weights as an optimization problem. MSOPSO adds the notion of a controllable search space for the velocity weights to the concepts of SOPSO. RSOPSO adds the concept of an attractive and a repulsive phase to the concepts of SOPSO. MRSOPSO combines the concepts of SOPSO, the controllable search space for the velocity weights of MSOPSO, and the attractive and repulsive phase of RSOPSO.

We combined the four proposed APSO variants with the four known PSO variants GBPSO, DW-PSO, TVACPSO, and GCPSO into one PSO software package. Experiments using this software package showed that the proposed APSO variants outperform the implemented known PSO variants for high numbers of function evaluations. For lower numbers of function evaluations, the performances are similar. For low numbers of function evaluations, the known PSO variants outperform the proposed APSO variants. These results are supported by experiments on higher-dimensional problems.

5.1 Contributions

The following concepts have been developed in this thesis. First, the concept of optimizing the velocity weights. This optimization of the velocity weights includes the proposed objective function for the velocity weights; i.e., Equation (3.2), the proposed normalization employed to the improvements in the objective values as shown in Equations (3.3), and the proposed concept of taking one step of optimizing the velocity weights after every iteration. In addition, we developed the MSOPSO concept of using moving bounds with the optimization of the velocity weights as in Equations (3.5), (3.6), and (3.7). We further proposed the RSOPSO concept of combining the optimization of the velocity weights with the concept of an attractive and a repulsive phase as used in the attractive-repulsive PSO [56]. This concept includes the use of the mean separations $s^{(i)}(n)$ as objective value of the velocity weights in the repulsive phase as shown in Equation (3.9). The RSOPSO concept of setting the mean separation absolute upper threshold $s_u(n)$ to the mean sepa-

ration after initialization of the positions and setting the mean separation absolute lower threshold $s_l(n)$ to the mean separation absolute upper threshold $s_u(n)$ divided by a set divisor, is novel to the best of our knowledge. Additionally, RSOPSO proposes using a mean separation absolute lower \check{s}_l and upper divisor \check{s}_u to adaptively change the mean separation absolute lower $s_l(n)$ and upper threshold $s_u(n)$ as shown in Equations (3.10). MRSOPSO combines the optimization of the velocity weights from SOPSO, moving bounds as employed for MSOPSO, and attractive and repulsive phases as employed for RSOPSO. Further, MRSOPSO introduces the normalization of the mean separation $s(n)$ relative to the mean separation absolute lower $s_l(n)$ and upper threshold $s_u(n)$ as shown in Equation (3.11).

For using the PSO software package we make the following recommendations. For difficult problems that require high numbers of function evaluations such as one to several million we recommend SOPSO, for problems that require lower numbers of function evaluations such as few hundred thousand we recommend GBPSO and SOPSO, for problems that require low numbers of function evaluations such as few thousand we recommend TVACPSO, and for problems with no good estimate on the required number of function evaluations we recommend SOPSO and high numbers of function evaluations.

5.2 Further Work

The parameters of the four proposed APSO variants have been empirically optimized. We found good, general-purpose settings for the proposed APSO variants by empirically trying different settings. Settings that performed well for all 22 test problems were selected. The settings were optimized with focus on good performance for high numbers of function evaluations. Not all possible parameter settings could be evaluated. Therefore, it might be possible to find better settings but finding such settings is a difficult task.

The further work includes the comparison of the PSO software package to other optimization software packages, such as LGO [49] and VTDirect [17]. To make the MSOPSO, RSOPSO, and MRSOPSO variants faster, the mean separation $s(n)$ could be replaced by a less expensive indicator of the distribution of the positions in the search space. For example, such an indicator could be computed by looking at the separation between randomly selected particles instead of looking at the separation between all particles. The change of the indicator would need to be accompanied by experiments to determine whether it changes the performance of RSOPSO or any other SOPSO variant using that indicator. Further, we propose the addition of parallel PSO techniques to the software package.

REFERENCES

- [1] B. Blaha and D. Wunsch. Evolutionary programming to optimize an assembly program. *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02*, 2:1901–1903, 2002.
- [2] Y. Bo, Z. Ding-Xue, and L. Rui-Quan. A modified particle swarm optimization algorithm with dynamic adaptive. *2007 Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2:346–349, 2007.
- [3] X. Cai, Z. Cui, J. Zeng, and Y. Tan. Self-learning particle swarm optimization based on environmental feedback. *Innovative Computing, Information and Control, 2007. ICICIC '07*, page 570, 2007.
- [4] L. Chan, J. Karceski, and J. Lakonishok. On portfolio optimization: forecasting covariances and choosing the risk model. *Review of Financial Studies*, 12(5):2937–2974, 1999.
- [5] S.H. Chen, J. Wu, and Y.D. Chen. Interval optimization for uncertain structures. *Finite Elements in Analysis and Design*, 40(11):1379–1398, 2004.
- [6] Y. Choi and H. G. Stenger. Kinetics, simulation and optimization of methanol steam reformer for fuel cell applications. *Journal of Power Sources*, 142(1-2):81–91, 2005.
- [7] J. Chuanwen and E. Bompard. A self-adaptive chaotic particle swarm algorithm for short term hydroelectric system scheduling in deregulated environment. *Energy Conversion and Management*, 46(17):2689–2696, 2005.
- [8] M. Clerc. Semi-continuous challenge, http://clerc.maurice.free.fr/pso/semi-continuous_challenge/, April 2004.
- [9] J. Digalakis and K. Margaritis. Performance comparison of memetic algorithms. *Applied Mathematics and Computation (New York)*, 158(1):237–252, 2004.
- [10] L. dos Santos Coelho and V.C. Mariani. An efficient particle swarm optimization approach based on cultural algorithm applied to mechanical design. *Evolutionary Computation, 2006. CEC 2006*, pages 1099–1104, 2006.
- [11] S. Dykes, R.G. Dykes, and B.E. Rosen. Parallel very fast simulated reannealing by temperature block partitioning. In *IEEE International Conference on Systems, Man, and Cybernetics. Humans, Information and Technology*, pages 1914–1919. IEEE press, 1994.
- [12] A. Eidehall and L. Petersson. Threat assessment for general road scenes using Monte Carlo sampling. *2006 IEEE Intelligent Transportation Systems Conference*, 2006.
- [13] A. Engelbrecht. *Computational Intelligence — An Introduction 2nd Edition*. Wiley, 2007.
- [14] F. Gao, Q. Zhao, H. Liu, and G. Cui. Cultural particle swarm algorithms for constrained multi-objective optimization. *ICCS '07: Proceedings of the 7th International Conference on Computational Science, Part IV*, pages 1021–1028, 2007.
- [15] Y. Gao and Y. Duan. A new particle swarm optimization algorithm with adaptive mutation operator. *The Second International Conference on Information and Computing Science, ICIC 2009*, 1:58–61, 2009.

- [16] X. Geng, J. Xu, J. Xiao, and L. Pan. A simple simulated annealing algorithm for the maximum clique problem. *Information Sciences*, 177(22):5064–5071, 2007.
- [17] J. He, L.T. Watson, and M. Sosenkina. VTDirect95, <http://www.scl.ameslab.gov/~masha/vtdirect95/>, November 2010.
- [18] Q. He and L. Wang. An effective co-evolutionary particle swarm optimization for constrained engineering design problems. *Engineering Applications of Artificial Intelligence*, 20(1):89–99, 2007.
- [19] A.R. Hedar. GO test problems, http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/hedar_files/testgo_files/page364.htm, 2011.
- [20] M. Holena. Genetic algorithms for the optimization of catalysts in chemical engineering. *Proceedings of the World Congress on Engineering and Computer Science*, 2008.
- [21] X. Hu and R.C. Eberhart. Adaptive particle swarm optimization: detection and response to dynamic systems. *CEC '02: Proceedings of the Evolutionary Computation on 2002*, pages 1666–1670, 2002.
- [22] A. Ide and K. Yasuda. A basic study of adaptive particle swarm optimization. *Denki Gakkai Ronbunshi / Electrical Engineering in Japan*, 151(3):41–49, 2005.
- [23] Frontline Systems Inc. Optimization problem types, <http://www.solver.com/probtype.htm>, 2010.
- [24] N. Iwasaki and K. Yasuda. Adaptive particle swarm optimization via velocity feedback. *Proceedings of the 36th ISCIE International Symposium on Stochastic Systems Theory and its Applications*, pages 357–362, 2005.
- [25] K. Izui, S. Nishiwaki, M. Yoshimura, M. Nakamura, and J.E. Renaud. Enhanced multiobjective particle swarm optimization in combination with adaptive weighted gradient-based searching. *Taylor & Francis LTD*, 2008.
- [26] M.T. Jonsson and T.P. Runarsson. Optimizing the sailing route for fixed groundfish survey stations. *International Council for the Exploration of the Sea*, 1996.
- [27] J. Jordan, S. Helwig, and R. Wanka. Social interaction in particle swarm optimization, the ranked fips, and adaptive multi-swarms. *GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pages 49–56, 2008.
- [28] J. Kennedy and R. Eberhart. Particle swarm optimization. *Proceedings, IEEE International Conference on Neural Networks*, 4:1942–1948, 1995.
- [29] J. Kennedy and R. Mendes. Neighborhood topologies in fully-informed and best-of-neighborhood particle swarms. *SMCia/03. Proceedings of the 2003 IEEE International Workshop on Soft Computing in Industrial Applications*, pages 45–50, 2003.
- [30] B. Koh, A.D. George, R.T. Haftka, and B.J. Fregly. Parallel asynchronous particle swarm optimization. *International Journal for Numerical Methods in Engineering*, 67(4):578–595, 2006.
- [31] G. Lapizco-Encinas, C. Kingsford, and J. Reggia. A cooperative combinatorial particle swarm optimization algorithm for side-chain packing. *2009 IEEE Swarm Intelligence Symposium*, 2009.
- [32] X. Li, H. Fu, and C. Zhang. A self-adaptive particle swarm optimization algorithm. *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, pages 186–189, 2008.

- [33] Z. Li-Ping, Y. Huan-Jun, and H. Shang-Xu. Optimal choice of parameters for particle swarm optimization. *Journal of Zhejiang University - Science A*, 6:528–534, 2005.
- [34] M. Lin and J. Wawrzyniek. Improving FPGA placement with dynamically adaptive stochastic tunneling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(12):1858–1869, 2010.
- [35] H. Liu. Fuzzy adaptive turbulent particle swarm optimization. *Proceedings IEEE Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, 2005.
- [36] S. Liu and J. Wang. An improved self-adaptive particle swarm optimization approach for short-term scheduling of hydro system. *2009 International Asia Conference on Informatics in Control, Automation and Robotics. CAR 2009*, pages 334–338, 2009.
- [37] R. Luus and B. Bojkov. Global optimization of the bifunctional catalyst problem. *The Canadian Journal of Chemical Engineering*, (1):160–163, 2009.
- [38] J. Machta. Strengths and weaknesses of parallel tempering. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 80(5), 2009.
- [39] M. Meissner. PsoVis short introduction, <http://gecco.org.chemie.uni-frankfurt.de/psovis/documentation.html>, 2011.
- [40] M. Meissner, M. Schmuker, and G. Schneider. Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics*, 7(1):125, 2006.
- [41] M. Molga and C. Smutnicki. Test functions for optimization needs, <http://zsd.iar.pwr.wroc.pl/>, 2005.
- [42] F. Muyl, L. Dumas, and V. Herbert. Hybrid method for aerodynamic shape optimization in automotive industry. *Computers and Fluids*, 33(5-6):849–858, 2004.
- [43] T. Niknam. A new fuzzy adaptive hybrid particle swarm optimization algorithm for non-linear, non-smooth and non-convex economic dispatch problem. *Applied Energy*, 87(1):327–339, 2010.
- [44] B. Niua, Y. Zhua, X. Hea, and H. Wuc. MCPSO: A multi-swarm cooperative particle swarm optimizer. *Applied Mathematics and Computation*, 185(2):1050–1062, 2007.
- [45] M.G.H. Omran, A. Engelbrecht, and A. Salman. Barebones particle swarm for integer programming problems. *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 170–175, 2007.
- [46] M.G.H. Omran, A.P. Engelbrecht, and A. Salman. Differential evolution based particle swarm optimization. *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 112–119, 2007.
- [47] C. Pan, C. Xu, and G. Li. Differential evolutionary strategies for global optimization. *Shenzhen Daxue Xuebao (Ligong Ban) / Journal of Shenzhen University Science and Engineering*, 25(2):211–215, 2008.
- [48] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method in multiobjective problems. *SAC '02: Proceedings of the 2002 ACM symposium on applied computing*, pages 603–607, 2002.
- [49] J.D. Pintér. *LGO — A Model Development and Solver System for Continuous Global Optimization User Guide*. Pinter Consulting Services, 2005.
- [50] J.D. Pintér. Global optimization, <http://mathworld.wolfram.com/globaloptimization.html>, November 2010.
- [51] H. Pohlheim. GEATbx: Example functions (single and multi-objective functions) 2 parametric optimization, <http://www.geatbx.com/docu/fcnindex-01.html>, 2011.

- [52] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1:33–57, 2007.
- [53] A. Rakitianskaia and A.P. Engelbrecht. Cooperative charged particle swarm optimiser. *2008 IEEE Congress on Evolutionary Computation (CEC)*, pages 933–939, 2008.
- [54] C.R. Raquel and P.C. Naval, Jr. An effective use of crowding distance in multiobjective particle swarm optimization. *GECCO 2005 — Genetic and Evolutionary Computation Conference*, pages 257–264, 2005.
- [55] A. Ratnaweera, S.K. Halgamuge, and H.C. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transaction on Evolutionary Computation*, 8(3):240–255, 2004.
- [56] J. Riget and J.S. Vesterstrøm. A diversity-guided particle swarm optimizer — the ARPSO. *EVALife Technical Report no. 2002-2*, 2002.
- [57] S. Salmon. PSO Toolbox, <http://forge.scilab.org/>, 2011.
- [58] J. Schutte, J.A. Reinbolt, B.J. Fregly, R.T. Haftka, and A.D. George. Parallel global optimization with the particle swarm algorithm. *International Journal for Numerical Methods in Engineering*, 61(13):2296–2315, 2004.
- [59] O. Schutze, E. Talbi, C.C. Coello, L.V. Santana-Quintero, and G.T. Pulido. A memetic pso algorithm for scalar optimization problems. *Proceedings of the 2007 IEEE Swarm Intelligence Symposium, SIS 2007*, pages 128–134, 2007.
- [60] Y. Shi and R. Eberhart. A modified particle swarm optimizer. *Proceedings of the IEEE World Congress on Computation Intelligence*, pages 69–73, 1998.
- [61] Y. Shi and R.A. Krohling. Co-evolutionary particle swarm optimization to solve min-max problems. *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*, pages 1682–1687, 2002.
- [62] J. Tang and X. Zhao. Particle swarm optimization with adaptive mutation. *2009 WASE International Conference on Information Engineering (ICIE)*, 2:234–237, 2009.
- [63] F. Van den Bergh and A.P. Engelbrecht. Cooperative learning in neural networks using particle swarm optimizers. *South African Computer Journal*, (26):84–90, 2000.
- [64] F. Van den Bergh and A.P. Engelbrecht. A new locally convergent particle swarm optimiser. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 3:94–99, 2002.
- [65] M. Veltman. Algebraic techniques. *Computer Physics Communications*, 3:75–78, September 1972.
- [66] R. Vetschera. A general branch-and-bound algorithm for fair division problems. *Computers and Operations Research*, 2010.
- [67] H. Wang, D. Wang, and S. Yang. A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. *Soft Computing*, 13(8):763–780, 2009.
- [68] M.Y. Wang, X. Wang, and D. Guo. A level set method for structural topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 192(1-2):227–246, 2003.
- [69] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *Evolutionary Computation*, 1, 1997.
- [70] D. Wong. Fortran data types and specification statements, <http://www-classes.usc.edu/engr/ce/108/text/fbk01.htm>, 2011.

- [71] C. Wu. Ant colony multilevel path optimize tactic based on information consistence optimize. *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, 1:533–536, 2010.
- [72] T. Yamaguchi, N. Iwasaki, and K. Yasuda. Adaptive particle swarm optimization using information about global best. *IEEJ Transactions on Electronics, Information and Systems*, 126:270–276, 2006.
- [73] T. Yamaguchi and K. Yasuda. Adaptive particle swarm optimization; self-coordinating mechanism with updating information. *IEEE International Conference on Systems, Man and Cybernetics, 2006. SMC '06*, 3:2303–2308, 2006.
- [74] Y. Yan and L.A. Osadciw. Density estimation using a new dimension adaptive particle swarm optimization algorithm. *Swarm Intelligence*, 3(4):275–301, 2009.
- [75] X. Yang and S. Deb. Cuckoo search via levy flights. *Proceedings of the 2009 World Congress on Nature; Biologically Inspired Computing (NaBIC 2009)*, pages 210 – 214, 2009.
- [76] X. Yang, J. Yuan, J. Yuan, and H. Mao. A modified particle swarm optimizer with dynamic adaptation. *Applied Mathematics and Computation*, 189(2):1205–1213, 2007.
- [77] K. Yao, F. Li, and X. Liu. An improved multi-particle swarm co-evolution algorithm. *2007 3rd International Conference on Natural Computation*, pages 51–55, 2007.
- [78] K. Yasuda, K. Yazawa, and M. Motoki. Particle swarm optimization with parameter self-adjusting mechanism. *IEEJ Transactions on Electrical and Electronic Engineering*, 5(2):256–257, 2010.
- [79] A.C. Zecchin, A.R. Simpson, H.R. Maier, and J.B. Nixon. Parametric study for an ant algorithm applied to water distribution system optimization. *IEEE Transactions on Evolutionary Computation*, 9(2):175–191, 2005.
- [80] Z.H. Zhan and J. Zhang. Adaptive particle swarm optimization. *Ant Colony Optimization and Swarm Intelligence. Proceedings 6th International Conference, ANTS 2008*, pages 227–234, 2008.
- [81] Z.H. Zhan, J. Zhang, Y. Li, and H.S.H. Chung. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(6):1362–1381, 2009.
- [82] J. Zhu, J. Zhao, and X. Li. A new adaptive particle swarm optimization algorithm. *International Workshop on Modelling, Simulation and Optimization*, pages 456–458, 2008.

APPENDIX A

PSO SOFTWARE PACKAGE PARAMETERS

The implemented PSO software package has the following input parameters (parameters are in capital letters in the style of Fortran coding practices):

1. `SEED` defines the random seed used.
2. `MAXIMIZATION` defines whether maximization or minimization is used. If set to true, the optimization problem to be solved is treated as a maximization problem. If set to false, the optimization problem to be solved is treated as a minimization problem.
3. `W_RANGE` is a two-dimensional array. The first value defines the inertia weight for the first iteration w_s , and the second value defines the inertia weight for the last iteration w_e , see Equation (2.4). Set $w_s = w_e$ for a constant inertia weight such as in GBPSO.
4. `C1_RANGE` is a two-dimensional array. The first value defines the personal best weight for the first iteration c_{1s} , and the second value defines the personal best weight for the last iteration c_{1e} , see Equation (2.6a). Set $c_{1s} = c_{1e}$ for a constant personal best weight such as in GBPSO.
5. `C2_RANGE` is a two-dimensional array. The first value defines the global best weight for the first iteration c_{2s} , and the second value defines the global best weight for the last iteration c_{2e} , see Equation (2.6b). Set $c_{2s} = c_{2e}$ for a constant global best weight such as in GBPSO.
6. `NPARTICLES` defines the number of particles n_p used, that is it defines the swarm size. Particles are numbered from 1 to n_p using i .
7. `MAXITERATIONS` defines the total number of iterations used. All PSO variants iterate from one to the maximum number of iterations.
8. `GUARANTEED_CONVERGENCE` defines whether Equation (2.7a) is used for calculating the velocity of the global best particle; i.e., whether guaranteed convergence is used. Set to true for using GCPSO.
9. `RHO` sets the search radius parameter ρ , see Equations (2.7).
10. `GC_FAILURE_NUMBER` defines the guaranteed convergence failure threshold a_c , see Equations (2.7).
11. `GC_SUCCESS_NUMBER` defines the guaranteed convergence success threshold s_c , see Equations (2.7).
12. `DIMENSIONS` specifies the dimensionality of positions.
13. `REDUCTION_FACTOR` defines the reduction factor α , see Equations (3.12).
14. `LOWER_BOUNDS` defines the lower bounds of the optimization problem.
15. `UPPER_BOUNDS` defines the upper bounds of the optimization problem.
16. `OBJECTIVE_DIMENSIONS` defines the dimensionality of the objective values returned by the objective function. This parameter is required to make it possible to add a PSO variation, capable of finding the Pareto front for multi-objective optimization problems, to our PSO software package.
17. `OBJECTIVE` defines the number of the objective that has to be optimized if only one objective from a multi-objective optimization problem has to be optimized.

18. `ADAPTIVE` defines whether adaptive concepts are used. Set to zero for using no adaptive concepts and to one for using the optimization of velocity weights.
19. `DISTRIBUTION` defines which distribution is used to initialize the positions of the velocity weights. Set to zero for using a Gaussian distribution and to one for using a uniform distribution.
20. `DISTRIBUTION_PARAMETERS` is a six-dimensional array that holds mean and variance values if a Gaussian distribution is used for initializing the positions of the velocity weights, and holds lower and upper bounds if a uniform distribution is used for initializing the positions of the velocity weights. If a uniform distribution is used, the first value in the array is the lower bound value for initializing the inertia weights $w^{(i)}$, the second value in the array is the upper bound value for initializing the inertia weights, the third value in the array is the lower bound value for initializing the personal best weights $c_1^{(i)}$, the fourth value in the array is the upper bound value for initializing the personal best weights, the fifth value in the array is the lower bound value for initializing the global best weights $c_2^{(i)}$, the sixth value in the array is the upper bound value for initializing the global best weights. If a Gaussian distribution is used, the first value in the array is the mean value for initializing the inertia weights $w^{(i)}$, the second value in the array is the variance for initializing the inertia weights, the third value in the array is the mean value for initializing the personal best weights $c_1^{(i)}$, the fourth value in the array is the variance for initializing the personal best weights, the fifth value in the array is the mean value for initializing the global best weights $c_2^{(i)}$, the sixth value in the array is the variance for initializing the global best weights.
21. `RANDOM_PERCENT` defines the percentage of particles that have their position of the velocity weights reinitialized randomly.
22. `AFTERX` defines after how many iterations velocity weights should be optimized.
23. `FORX` defines for how many iterations the velocity weights should be optimized.
24. `SCALE_POTENCY` defines a scale potency that is used in certain objective functions of the velocity weights.
25. `LOWER_BOUNDS_PARAMETERS` defines the lower bounds for optimizing the velocity weights. The first value in this array defines the lower bound for the inertia weights $w^{(i)}$, the second value the lower bound for the personal best weights $c_1^{(i)}$, and the third value the lower bound for the global best weights $c_2^{(i)}$.
26. `UPPER_BOUNDS_PARAMETERS` defines the upper bounds for optimizing the velocity weights. The first value in this array defines the upper bound for the inertia weights $w^{(i)}$, the second value the upper bound for the personal best weights $c_1^{(i)}$, and the third value the upper bound for the global best weights $c_2^{(i)}$.
27. `REDUCTION_FACTOR_PARAMETER` defines the reduction factor for the velocity weights $\tilde{\alpha}$.
28. `NUMBER_ADAPTIVE_PARAMETERS` defines the number of parameters that are adaptively changed (optimized) for every particle. Has to be set to three if using any proposed APSO variant.
29. `REINITIALIZE_PARAMETERS_BEST_AFTER` defines the number of iterations after which to reinitialize the personal best positions and the global best position of the velocity weights, and reset the personal best values and the global best values of the velocity weights. Is set to the total number of iterations for no reinitialization.
30. `WEIGHT_HISTORY_GLOBAL` defines the global weight factor \check{g} which is used to weigh the number of global best updates $u_g^{(i)}$ in certain objective functions of the velocity weights, see Equation (3.2).

31. `WEIGHT_HISTORY_LOCAL` defines the local weight factor \tilde{l} which is used to weigh the number of local best updates $u_p^{(i)}$ in certain objective functions for the velocity weights, see Equation (3.2).
32. `IMPROVEMENT_NORMALIZATION` defines how the differences in the objective values $d^{(i)}(n)$ are normalized between iterations. If set to zero, the differences in the objective values $d^{(i)}(n)$ are divided by the sum of the absolute value of all good and bad differences in the objective values. Where good means that the current objective value $f^{(i)}(n)$ is better than the previous objective value $f^{(i)}(n-1)$, and bad means that the current objective value $f^{(i)}(n)$ is worse than the previous objective value $f^{(i)}(n-1)$. If set to one, the differences in the objective values $d^{(i)}(n)$ are divided by the absolute value of the sum of all good differences in the objective values, see Equations (3.3). If set to three, the differences in the objective values $d^{(i)}(n)$ are divided by the sum of the absolute value of all differences in the objective values scaled by the iteration number. If set to four, the differences in the objective values $d^{(i)}(n)$ are divided by the sum of the absolute value of all good differences in the objective values scaled by the iteration number.
33. `PHASE_TYPE` sets whether different phases are used. If set to zero, no different phases are used. If set to one, a repulsive and an attractive phase such as described for RSOPSO are used; change in phases are detected based on the iteration count n ; and the attractive phase uses an improvement and history-based objective function for the velocity weights, whereas the repulsive phase uses a mean separation based objective function for the velocity weights. If set to two, a repulsive and an attractive phase are used; change in phases are detected based on the iteration count n ; and the attractive phase uses an improvement and history-based objective function for the velocity weights, whereas the repulsive phase sets the positions of the velocity weights to random negative values. If set to three, a repulsive and an attractive phase are used; change in phases are detected based on the mean separation $s(n)$ as calculated in Equations (3.8a); and the attractive phase uses an improvement and history-based objective function for the velocity weights, whereas the repulsive phase sets the positions of the velocity weights to random negative values. If set to four, a repulsive and an attractive phase are used; change in phases are detected based on the mean separation $s(n)$, see Equation (3.10); and the attractive phase uses an improvement and history-based objective function for the velocity weights, whereas the repulsive phase uses a mean separation based objective function for the velocity weights, see Equation (3.9).
34. `FIRST_PHASE_ITERATIONS` defines the number of iterations spent in the attractive phase before switching to the repulsive phase. This parameter is only used if phase changes are based on the iteration count n .
35. `SECOND_PHASE_ITERATIONS` defines the number of iterations spent in the repulsive phase before switching to the attractive phase. This parameter is only used if phase changes are based on the iteration count n .
36. `DISTRIBUTION_REINITIALIZATION` defines which kind of distribution is used to reinitialize positions of the velocity weights. Set to zero for using a Gaussian distribution and to one for using a uniform distribution.
37. `DISTRIBUTION_PARAMETERS_REINITIALIZATION` is a six-dimensional array that holds mean and variance values if a Gaussian distribution is used for reinitializing the positions of the velocity weights, and holds lower and upper bounds if a uniform distribution is used for reinitializing the positions of the velocity weights. If a uniform distribution is used, the first value in the array is the lower bound value for reinitializing the inertia weights $w^{(i)}$, the second value in the array is the upper bound value for reinitializing the inertia weights, the third value in the array is the lower bound value for reinitializing the personal best weights $c_1^{(i)}$, the fourth value in the array is the upper bound value for reinitializing the personal best weights, the fifth value in the array is the lower bound value for reinitializing the global

best weights $c_2^{(i)}$, the sixth value in the array is the upper bound value for reinitializing the global best weights. If a Gaussian distribution is used, the first value in the array is the mean value for reinitializing the inertia weights $w^{(i)}$, the second value in the array is the variance for reinitializing the inertia weights, the third value in the array is the mean value for reinitializing the personal best weights $c_1^{(i)}$, the fourth value in the array is the variance for reinitializing the personal best weights, the fifth value in the array is the mean value for reinitializing the global best weights $c_2^{(i)}$, the sixth value in the array is the variance for reinitializing the global best weights.

38. `CHANGE_MEAN` is a two-dimensional array. The first value sets the mean separation absolute lower threshold $s_l(n)$ and the second value sets the mean separation absolute upper threshold $s_u(n)$ as used in Equations (3.10). If the first value is set to a negative number, the mean separation absolute lower threshold $s_l(n)$ is set to the mean separation after initializing divided by the absolute value of the given negative number. If the second value is set to a negative number, the mean separation absolute upper threshold $s_u(n)$ is set to the mean separation after initializing the particles in the search space $s(0)$.
39. `CHANGE_MEAN_SCALE` is a two-dimensional array. The first value sets the mean separation absolute lower divisor \check{s}_l as used in Equation (3.10b) and the second value sets the mean separation absolute upper divisor \check{s}_u as used in Equation (3.10c).
40. `STOP_NUMBER_UNIMPROVED_PHASE_CYCLES` defines after how many phase cycles without improvement of the global best value \mathbf{x}_g , a PSO variants using phases such as RSOPSO has to stop. Is set to the total number of iterations if PSO variants should not stop before reaching the total number of iterations.
41. `BOUNDS_BY_TYPE` defines whether the bounds in which the velocity weights are optimized, are changed during optimization. If set to zero, bounds for optimizing the velocity weights are fixed and do not change. If set to one, moving bounds based on the mean separation $s(n)$ (Equation (3.8a)) and a uniform distribution for reinitializing the positions of the velocity weights are used. If set to two, moving bounds based on the mean separation and a Gaussian distribution for reinitializing the positions of the velocity weights are used. If set to three, moving bounds based on the percentage of remaining iterations $m(n)$ (Equation (3.9)) and a uniform distribution for reinitializing the positions of the velocity weights are used. If set to four, moving bounds based on the percentage of remaining iterations and a Gaussian distribution for reinitializing the positions of the velocity weights are used.
42. `W_BOUNDS_BY_MEAN` is an eight-dimensional array. If any PSO variation with moving bounds, but no different phases is used, only the first four values are used where the first value defines the absolute lower inertia weight \check{w}_l , the second value defines the absolute upper inertia weight \check{w}_u , the third value defines the inertia weight width \check{w}_w , and the fourth value defines the inertia weight bound flag \check{w}_f , see Equations (3.5). If any PSO variation with moving bounds and different phases is used, the first value defines the absolute lower inertia weight \check{w}_l used in the attractive phase, the second value defines the absolute upper inertia weight \check{w}_u used in the attractive phase, the third value defines the inertia weight width \check{w}_w used in the attractive phase, the fourth value defines the inertia weight bound flag \check{w}_f used in the attractive phase, the fifth value defines the absolute lower inertia weight \check{w}_l used in the repulsive phase, the sixth value defines the absolute upper inertia weight \check{w}_u in the repulsive phase, the seventh value defines the inertia weight width \check{w}_w used in the repulsive phase, and the eighth value defines the inertia weight bound flag \check{w}_f used in the repulsive phase, see Equations (3.5).
43. `C1_BOUNDS_BY_MEAN` is an eight-dimensional array. If any PSO variation with moving bounds, but no different phases is used, only the first four values are used where the first value defines the absolute lower personal best weight \check{c}_{1l} , the second value defines the absolute upper personal best weight \check{c}_{1u} , the third value defines the personal best weight width \check{c}_{1w} , and the fourth value defines the personal best weight bound flag \check{c}_{1f} , see Equations (3.6). If any PSO

variation with moving bounds and different phases is used, the first value defines the absolute lower personal best weight \check{c}_{1l} used in the attractive phase, the second value defines the absolute upper personal best weight \check{c}_{1u} used in the attractive phase, the third value defines the personal best weight width \check{c}_{1w} used in the attractive phase, the fourth value defines the personal best weight bound flag \check{c}_{1f} used in the attractive phase, the fifth value defines the absolute lower personal best weight \check{c}_{1l} used in the repulsive phase, the sixth value defines the absolute upper personal best weight \check{c}_{1u} used in the repulsive phase, the seventh value defines the personal best weight width \check{c}_{1w} used in the repulsive phase, and the eighth value defines the personal best weight bound flag \check{c}_{1f} used in the repulsive phase, see Equations (3.6).

44. `C2_BOUNDS_BY_MEAN` is an eight-dimensional array. If any PSO variation with moving bounds, but no different phases is used, only the first four values are used where the first value defines the absolute lower global best weight \check{c}_{2l} , the second value defines the absolute upper global best weight \check{c}_{2u} , the third value defines the global best weight width \check{c}_{2w} , and the fourth value defines the global best weight bound flag \check{c}_{2f} , see Equations (3.7). If any PSO variation with moving bounds and different phases is used, the first value defines the absolute lower global best weight \check{c}_{2l} used in the attractive phase, the second value defines the absolute upper global best weight \check{c}_{2u} used in the attractive phase, the third value defines the global best weight width \check{c}_{2w} used in the attractive phase, the fourth value defines the global best weight bound flag \check{c}_{2f} used in the attractive phase, the fifth value defines the absolute lower global best weight \check{c}_{2l} used in the repulsive phase, the sixth value defines the absolute upper global best weight \check{c}_{2u} used in the repulsive phase, the seventh value defines the global best weight width \check{c}_{2w} used in the repulsive phase, and the eighth value defines the global best weight bound flag \check{c}_{2f} used in the repulsive phase, see Equations (3.7).
45. `SOPSO_STYLE_GLOBAL_BEST_PARAMETER` defines how to choose the global best velocity weights. If set to true, uses the objective function for the velocity weights Equation (3.2) to determine the global best velocity weights. If set to false, sets the global best velocity weights to the velocity weights of the global best particle. Typically, set to true.
46. `MOVE_BACK_TYPE` defines how to move particles back into the search space if they try to leave it. Set to 0 for moving the particle recursively back using the reduction factor α with Equations (3.12) and to 1 for putting the particle to the bound in the dimension it violated the search space.
47. `MOVE_BACK_TYPE_PARAMETER` defines how to move velocity weights back into the search space for the velocity weights if they try to leave it. Set to 0 for moving the velocity weight recursively back using the reduction factor $\tilde{\alpha}$ with Equations (3.12) and to 1 for putting the velocity weights to the bound in the dimension they violated the search space for the velocity weights.
48. `DEBUG_LEVEL` defines what debug information is printed. Set to 0 for no debugging. Set to a value larger than 0 for printing the setting. Set to a value larger than 1 for printing the global best value after every iteration.

The PSO software package has the following output parameters:

1. `G_BEST_VALUE` the global best value found by the algorithm.
2. `G_BEST_POSITION` the global best position found by the algorithm.

APPENDIX B

SAMPLE CALL TO PSO SOFTWARE PACKAGE

PROGRAM mainSerial

```
INTEGER , PARAMETER :: NUMBER_ADAPTIVE_PARAMETERS = 3
INTEGER :: SEED, NPARTICLES, MAXITERATIONS, OBJECTIVE_DIMENSIONS, I,
OBJECTIVE, FUNCTION_EVALUATIONS, AFTERX, FORX,
REINITIALIZE_PARAMETERS_BEST_AFTER, IMPROVEMENT_NORMALIZATION,
DIMENSIONS, PHASE_TYPE, FIRST_PHASE_ITERATIONS, SECOND_PHASE_ITERATIONS,
DISTRIBUTION, DISTRIBUTION_REINITIALIZATION,
STOP_NUMBER_UNIMPROVED_PHASE_CYCLES, BOUNDS_BY_TYPE, GC_FAILURE_NUMBER,
GC_SUCCESS_NUMBER, MOVE_BACK_TYPE, MOVE_BACK_TYPE_PARAMETER, END_TIME(8)
, START_TIME(8), TIME_NEEDED, DEBUG_LEVEL
INTEGER(2) :: io_status

CHARACTER (LEN = 10) CHARS(3)

LOGICAL :: MAXIMIZATION, GUARANTEED_CONVERGENCE, ADAPTIVE,
SOPSO_STYLE_GLOBAL_BEST_PARAMETER

DOUBLE PRECISION :: REDUCTION_FACTOR, REDUCTION_FACTOR_PARAMETER,
G_BEST_VALUE, RANDOM_PERCENT, SCALE_POTENCY, WEIGHT_HISTORY_GLOBAL, RHO,
WEIGHT_HISTORY_LOCAL, CHANGE_MEAN(2), CHANGE_MEAN_SCALE(2), W_RANGE(2),
C1_RANGE(2), C2_RANGE(2), DISTRIBUTION_PARAMETERS(
NUMBER_ADAPTIVE_PARAMETERS*2), LOWER_BOUNDS_PARAMETERS(
NUMBER_ADAPTIVE_PARAMETERS), UPPER_BOUNDS_PARAMETERS(
NUMBER_ADAPTIVE_PARAMETERS), DISTRIBUTION_PARAMETERS_REINITIALIZATION(
NUMBER_ADAPTIVE_PARAMETERS*2), W_BOUNDS_BY_MEAN(8), C1_BOUNDS_BY_MEAN(8)
, C2_BOUNDS_BY_MEAN(8)
DOUBLE PRECISION , ALLOCATABLE :: LOWER_BOUNDS(:), UPPER_BOUNDS(:),
G_BEST_POSITION(:)

DIMENSIONS = 30
ALLOCATE(LOWER_BOUNDS(DIMENSIONS), UPPER_BOUNDS(DIMENSIONS), G_BEST_POSITION
(DIMENSIONS))
SEED = 55
NPARTICLES = 100
MAXITERATIONS = 74999
OBJECTIVE_DIMENSIONS = 1
OBJECTIVE = 1
REDUCTION_FACTOR = 0.54d0
LOWER_BOUNDS = -10.0d0
UPPER_BOUNDS = 10.0d0
W_RANGE(1) = 0.7298d0
W_RANGE(2) = 0.7298d0
C1_RANGE(1) = 1.49618d0
C1_RANGE(2) = 1.49618d0
C2_RANGE(1) = 1.49618d0
C2_RANGE(2) = 1.49618d0
MAXIMIZATION = .False.
GUARANTEED_CONVERGENCE = .False.
RHO = 1.0d0
GC_FAILURE_NUMBER = 5
GC_SUCCESS_NUMBER = 15
ADAPTIVE = .False.
DISTRIBUTION = 1
DISTRIBUTION_PARAMETERS(1) = 0.4d0
DISTRIBUTION_PARAMETERS(2) = 0.9d0
DISTRIBUTION_PARAMETERS(3) = 0.5d0
DISTRIBUTION_PARAMETERS(4) = 2.5d0
DISTRIBUTION_PARAMETERS(5) = 0.5d0
DISTRIBUTION_PARAMETERS(6) = 2.5d0
```

```

RANDOMPERCENT = 0.33 d0
AFTERX = 1
FORX = 1
SCALE.POTENCY = 0.5 d0
LOWER_BOUNDS.PARAMETERS(1) = -0.5 d0
UPPER_BOUNDS.PARAMETERS(1) = 2.0 d0
LOWER_BOUNDS.PARAMETERS(2) = -1.0 d0
UPPER_BOUNDS.PARAMETERS(2) = 4.2 d0
LOWER_BOUNDS.PARAMETERS(3) = -1.0 d0
UPPER_BOUNDS.PARAMETERS(3) = 4.2 d0
REDUCTION_FACTOR.PARAMETER = 0.50 d0
WEIGHT_HISTORY_GLOBAL = 6.0 d0
WEIGHT_HISTORY_LOCAL = 1.0 d0
REINITIALIZE.PARAMETERS.BEST_AFTER = 50
IMPROVEMENT.NORMALIZATION = 1
PHASE.TYPE = 0
FIRST.PHASE.ITERATIONS = 36
SECOND.PHASE.ITERATIONS = 2
DISTRIBUTION.REINITIALIZATION = 1
DISTRIBUTION.PARAMETERS.REINITIALIZATION(1) = 0.5 d0
DISTRIBUTION.PARAMETERS.REINITIALIZATION(2) = 0.8 d0
DISTRIBUTION.PARAMETERS.REINITIALIZATION(3) = 0.6 d0
DISTRIBUTION.PARAMETERS.REINITIALIZATION(4) = 2.4 d0
DISTRIBUTION.PARAMETERS.REINITIALIZATION(5) = 0.6 d0
DISTRIBUTION.PARAMETERS.REINITIALIZATION(6) = 2.4 d0
CHANGE.MEAN(1) = -100.0 d0
CHANGE.MEAN.SCALE(1) = 10.0 d0
CHANGE.MEAN(2) = -1.0 d0
CHANGE.MEAN.SCALE(2) = 2.5 d0
STOP.NUMBER.UNIMPROVED.PHASE.CYCLES = 40
BOUNDS.BY.TYPE = 0
W.BOUNDS.BY.MEAN(1) = 0.3 d0
C1.BOUNDS.BY.MEAN(1) = 0.5 d0
C2.BOUNDS.BY.MEAN(1) = 0.6 d0
W.BOUNDS.BY.MEAN(2) = 0.9 d0
C1.BOUNDS.BY.MEAN(2) = 2.5 d0
C2.BOUNDS.BY.MEAN(2) = 2.4 d0
W.BOUNDS.BY.MEAN(3) = 0.2 d0
C1.BOUNDS.BY.MEAN(3) = 0.2 d0
C2.BOUNDS.BY.MEAN(3) = 0.2 d0
W.BOUNDS.BY.MEAN(4) = -1.0 d0
C1.BOUNDS.BY.MEAN(4) = -1.0 d0
C2.BOUNDS.BY.MEAN(4) = 1.0 d0
W.BOUNDS.BY.MEAN(5) = 0.3 d0
C1.BOUNDS.BY.MEAN(5) = -2.5 d0
C2.BOUNDS.BY.MEAN(5) = -2.5 d0
W.BOUNDS.BY.MEAN(6) = 0.9 d0
C1.BOUNDS.BY.MEAN(6) = -0.5 d0
C2.BOUNDS.BY.MEAN(6) = -0.5 d0
W.BOUNDS.BY.MEAN(7) = 0.2 d0
C1.BOUNDS.BY.MEAN(7) = 0.2 d0
C2.BOUNDS.BY.MEAN(7) = 0.2 d0
W.BOUNDS.BY.MEAN(8) = -1.0 d0
C1.BOUNDS.BY.MEAN(8) = 1.0 d0
C2.BOUNDS.BY.MEAN(8) = -1.0 d0
SOPSO.STYLE.GLOBAL.BEST.PARAMETER = .True.
MOVE.BACK.TYPE = 0
MOVE.BACK.TYPE.PARAMETER = 0
DEBUG.LEVEL = 0

```

```

CALL DATE.AND.TIME(CHARS(1) , CHARS(2) , CHARS(3) , START.TIME)

```

```

CALL runPsoSerial(SEED, MAXIMIZATION, W.RANGE, C1.RANGE, C2.RANGE,
NPARTICLES, MAXITERATIONS, GUARANTEED.CONVERGENCE, RHO,
GC.FAILURE.NUMBER, GC.SUCCESS.NUMBER, DIMENSIONS, LOWER.BOUNDS,
UPPER.BOUNDS, REDUCTION.FACTOR, OBJECTIVE.DIMENSIONS, OBJECTIVE,
ADAPTIVE, DISTRIBUTION, DISTRIBUTION.PARAMETERS, RANDOMPERCENT
, AFTERX, FORX, SCALE.POTENCY, LOWER.BOUNDS.PARAMETERS,

```

```

        UPPER_BOUNDS.PARAMETERS, REDUCTION_FACTOR.PARAMETER,
        NUMBER_ADAPTIVE.PARAMETERS, REINITIALIZE.PARAMETERS.BEST_AFTER,
        WEIGHT_HISTORY.GLOBAL, WEIGHT_HISTORY.LOCAL,
        IMPROVEMENT.NORMALIZATION, PHASE.TYPE, FIRST_PHASE.ITERATIONS,
        SECOND_PHASE.ITERATIONS, DISTRIBUTION.REINITIALIZATION,
        DISTRIBUTION.PARAMETERS.REINITIALIZATION, CHANGE_MEAN,
        CHANGE_MEAN.SCALE, STOP_NUMBER.UNIMPROVED.PHASE.CYCLES,
        BOUNDS_BY.TYPE, W.BOUNDS_BY.MEAN, C1.BOUNDS_BY.MEAN,
        C2.BOUNDS_BY.MEAN, SOPSO.STYLE.GLOBAL.BEST.PARAMETER,
        MOVE.BACK.TYPE, MOVE.BACK.TYPE.PARAMETER, DEBUG.LEVEL,
        G.BEST.VALUE, G.BEST.POSITION)

    CALL DATE_AND_TIME(CHARS(1), CHARs(2), CHARs(3), END.TIME)

    FUNCTION_EVALUATIONS = NPARTICLES * (MAXITERATIONS + 1)

    OPEN(UNIT=103, FILE='result.txt', STATUS='NEW', IOSTAT=io_status)

    WRITE (UNIT=103, FMT=100) G.BEST.VALUE
    WRITE (UNIT=103, FMT=105) FUNCTION_EVALUATIONS
    WRITE (UNIT=103, FMT=101)

        DO I = 1, DIMENSIONS
            WRITE (UNIT=103, FMT=102) G.BEST.POSITION(I)
        END DO

    CALL COMPUTE.TIME(TIME.NEEDED, END.TIME, START.TIME)

    WRITE(UNIT=103, FMT=103) TIME.NEEDED

100    FORMAT('gBestValue: ', F50.20)
101    FORMAT('position Vector: ')
102    FORMAT(F50.20)
103    FORMAT('time required: ', I30)
105    FORMAT('functionEvaluations: ', I50)

        CLOSE(UNIT=103)

    DEALLOCATE(LOWER_BOUNDS, UPPER_BOUNDS, G.BEST.POSITION)

END

```

APPENDIX C

TEST FUNCTION CONFIGURATIONS

The values of vector $\check{\mathbf{a}}$ of De Jong 5, Equation (4.4), are set to:

$$\begin{aligned}\check{a}_1, \check{a}_6, \check{a}_{11}, \check{a}_{16}, \check{a}_{21} &= 32 \\ \check{a}_2, \check{a}_7, \check{a}_{12}, \check{a}_{17}, \check{a}_{22} &= 16 \\ \check{a}_3, \check{a}_8, \check{a}_{13}, \check{a}_{18}, \check{a}_{23} &= 0 \\ \check{a}_4, \check{a}_9, \check{a}_{14}, \check{a}_{19}, \check{a}_{24} &= 16 \\ \check{a}_5, \check{a}_{10}, \check{a}_{15}, \check{a}_{20}, \check{a}_{25} &= 32\end{aligned}$$

The values of vector $\check{\mathbf{b}}$ of De Jong 5, Equation (4.4), are set to:

$$\begin{aligned}\check{b}_1, \check{b}_2, \check{b}_3, \check{b}_4, \check{b}_5 &= 32 \\ \check{b}_6, \check{b}_7, \check{b}_8, \check{b}_9, \check{b}_{10} &= 16 \\ \check{b}_{11}, \check{b}_{12}, \check{b}_{13}, \check{b}_{14}, \check{b}_{15} &= 0 \\ \check{b}_{16}, \check{b}_{17}, \check{b}_{18}, \check{b}_{19}, \check{b}_{20} &= 16 \\ \check{b}_{21}, \check{b}_{22}, \check{b}_{23}, \check{b}_{24}, \check{b}_{25} &= 32\end{aligned}$$

The values of vector $\check{\mathbf{c}}$ of Deceptive Type 3, Equation (4.5), are set to:

$$\check{\mathbf{c}} = \begin{pmatrix} 0.56 \\ 0.65 \\ 0.12 \\ 0.98 \\ 0.44 \\ 0.34 \\ 0.59 \\ 0.79 \\ 0.88 \\ 0.04 \\ 0.25 \\ 0.81 \\ 0.30 \\ 0.49 \\ 0.53 \\ 0.21 \\ 0.61 \\ 0.86 \\ 0.31 \\ 0.29 \\ 0.84 \\ 0.72 \\ 0.92 \\ 0.77 \\ 0.39 \\ 0.11 \\ 0.01 \\ 0.03 \\ 0.43 \\ 0.80 \end{pmatrix}$$

APPENDIX D

RESULT TABLES CONSIDERING AVERAGE SOLUTIONS

Table D.1: Count of wins, draws, and losses for 7,500,000 FE considering average solutions.

Solver	win	draw	loss
GBPSO	0	16	6
DWPSO	0	17	5
TVACPSO	1	17	4
GCPSO	0	17	5
SOPSO	0	20	2
MSOPSO	0	16	6
RSOPSO	0	20	2
MRSOPSO	0	17	5

Table D.2: Count of wins, draws, and losses for 1,500,000 FE considering average solutions.

Solver	win	draw	loss
GBPSO	1	15	6
DWPSO	0	16	6
TVACPSO	0	14	8
GCPSO	0	16	6
SOPSO	0	18	4
MSOPSO	0	14	8
RSOPSO	1	18	3
MRSOPSO	1	16	5

Table D.3: Count of wins, draws, and losses for 250,000 FE considering average solutions.

Solver	win	draw	loss
GBPSO	3	12	7
DWPSO	0	14	8
TVACPSO	0	14	8
GCPSO	0	14	8
SOPSO	0	15	7
MSOPSO	2	12	8
RSOPSO	0	15	7
MRSOPSO	0	12	10

Table D.4: Count of wins, draws, and losses for 3,000 FE considering average solutions.

Solver	win	draw	loss
GBPSO	1	1	20
DWPSO	1	3	18
TVACPSO	8	5	9
GCPSO	1	3	18
SOPSO	1	5	16
MSOPSO	3	1	18
RSOPSO	0	5	17
MRSOPSO	1	4	17

Table D.5: PSO variants vs. APDO variants for 7,500,000 FE considering average solutions.

Test Problem	known PSO variants	proposed APDO variants
Griewank	0.0024653468	0.0057470070
Michalewicz	0.0016370493	0.0
Non-continuous Rastrigin	0.3355457426	0.0
Rastrigin	4.6431422664	0.3316530190
Rosenbrock	0.7611433973	0.0

Table D.6: PSO variants vs. APSO variants for 1,500,000 FE considering average solutions.

Test Problem	known PSO variants	proposed APSO variants
Griewank	0.0090172271	0.0032857615
Michalewicz	0.0141742391	0.0139209371
Non-continuous Rastrigin	3.3333333365	0.6666666667
Parabola	0.0000000246	0.0003476957
Rastrigin	11.2761992885	7.6280194377
Rosenbrock	2.2940880387	1.3388943063

Table D.7: PSO variants vs. APSO variants for 250,000 FE considering average solutions.

Test Problem	known PSO variants	proposed APSO variants
Ackley	0.0	0.0000000007
Griewank	0.0114923776	0.0114809353
Hyper-ellipsoid	0.0000004069	0.0050324897
Michalewicz	0.0294789235	0.0
Non-continuous Rastrigin	11.3333333333	10.6666666667
Parabola	9.9445068671	7.8283158568
Rastrigin	17.3599442294	14.6057951797
Rosenbrock	15.9563162600	19.3760885214
Sphere	0.0000055749	0.0042017392

Table D.8: PSO variants vs. APSO variants for 3,000 FE considering average solutions.

Test Problem	known PSO variants	proposed APSO variants
Ackley	2.3471185663	2.8503274229
Alpine	0.0410857362	0.0019377298
Drop Wave	0.0000003024	0.0
Generalized Penalized	8.1764346980	15.0314828781
Griewank	1.1419446639	1.3269625993
Hyper-ellipsoid	1491.2270736500	1751.5966941000
Michalewicz	1.0209907626	1.0428209165
Non-continuous Rastrigin	87.6822596283	64.0808451603
Parabola	2773.0872997900	2670.2379967400
Rastrigin	121.5536371850	162.9028143630
Rosenbrock	416.6967614490	338.5360761200
Schaffer F6	0.0032388438	0.0066067998
Schwefel P2.22	4.8289577581	5.0141165042
Shubert	0.0000000044	0.0000000128
Sphere	16484.4946035000	14264.7843282000
Step	90.0	188.3333333330
Tripod	0.0000749267	0.6666788690

Table D.9: Proposed APSO comparison for 7,500,000 FE considering average solutions.

Test Problem	SOPSO	MSOPSO	RSOPSO	MRSOPSO
Griewank	0.0057511083	0.0057470070	0.0057511083	0.0235910931
Michalewicz	0.0	0.1032496660	0.0	0.0149194455
Non-continuous Rastrigin	0.0	1.0	0.0	11.0
Parabola	0.0000000089	0.0000044325	0.0000000089	0.0
Rastrigin	0.3316530190	1.9899181142	0.3316530190	8.9546264760
Rosenbrock	0.0	11.1545012133	0.0	15.9014241671

Table D.10: Proposed APSO comparison for 1,500,000 FE considering average solutions.

Test Problem	SOPSO	MSOPSO	RSOPSO	MRSOPSO
Generalized Penalized	0.0345563401	0.0	0.0345563401	0.0
Griewank	0.0131250469	0.0114850364	0.0131250469	0.0032857615
Hyper-ellipsoid	0.0	0.0000002060	0.0	0.0
Michalewicz	0.0139209371	0.0288403826	0.0139209371	0.0981156218
Non-continuous Rastrigin	0.6666666667	5.6666666667	0.6666666667	7.6666666667
Parabola	0.0003476957	0.0494831227	0.0003476957	0.0011841635
Rastrigin	8.2913254758	13.2661123649	7.6280194377	12.2711566457
Rosenbrock	1.3388943063	13.7875290787	1.3388943063	14.0506079977
Sphere	0.0	0.0000000160	0.0	0.0

Table D.11: Proposed APSO comparison for 250,000 FE considering average solutions.

Test Problem	SOPSO	MSOPSO	RSOPSO	MRSOPSO
Ackley	0.3850506364	0.0000000007	0.3850506364	0.0000001240
Generalized Penalized	0.1036546596	0.0	0.1036546596	0.0
Griewank	0.0114809353	0.0278590931	0.0114809353	0.0252899560
Hyper-ellipsoid	0.0050324897	0.0530336489	0.0050324897	0.0108978670
Michalewicz	0.0	0.1686187550	0.0	0.1858121646
Non-continuous Rastrigin	10.6666666667	12.4595934528	10.6666666667	14.3333333333
Parabola	60.4532748691	7.8283158568	43.3526420540	19.5380423103
Rastrigin	27.8594261633	14.6057951797	27.8594261633	18.5725813378
Rosenbrock	19.3760885214	23.8375488671	19.3760885214	22.9722357465
Schwefel P2.22	0.0	0.0000000243	0.0	0.0000001654
Sphere	11.0227659992	1.3606351705	11.0227659992	0.0042017392

Table D.12: Proposed APSO comparison for 3,000 FE considering average solutions.

Test Problem	SOPSO	MSOPSO	RSOPSO	MRSOPSO
Ackley	4.2259047998	2.8503274229	4.2259047998	2.9884090174
Alpine	0.0019377298	0.0272875912	0.0032733162	0.0459792947
Camel Back	0.0	0.0000000007	0.0	0.0
De Jong 5	0.0	0.0000000055	0.0	0.0
Drop Wave	0.0	0.0000002114	0.0	0.0212515714
Easom	0.0100008239	0.0000002062	0.0100008239	0.0
Gen... Penalized	1552.9890236400	15.0314828781	1552.9890236400	15.6108038420
Griewank	2.4082685107	1.3269625993	2.4082685107	1.3795156439
Goldstein-Price	0.0	0.0000000022	0.0	0.0000000063
Hyper-ellipsoid	3013.8873529400	1751.5966941000	3013.8873529400	2353.1607630600
Michalewicz	1.4705368726	1.0428209165	1.4709665130	1.2753618142
Non... Rastrigin	87.9851645311	64.0808451603	87.9851645311	105.1007572380
Parabola	3538.5208554600	2670.2379967400	3538.5208554600	3243.1342495000
Rastrigin	197.4109640190	191.6116738170	197.4109640190	162.9028143630
Rosenbrock	2407.0916692800	1484.7810176200	2407.0916692800	338.5360761200
Schaffer F6	0.0066067998	0.0083638294	0.0066067998	0.0067012035
Schweffel P2.22	12.8455019338	5.0141165042	12.8455019338	13.2568263189
Shubert	0.0079415086	0.0000003066	0.0079415088	0.0000000128
Sphere	20253.4693065000	14264.7843282000	20253.4693065000	15926.9121206000
Step	889.3333333330	188.3333333330	889.3333333330	247.6666666670
Tripod	0.6666808227	1.3336440366	0.6670358543	0.6666788690

Table D.13: PSO comparison for 7,500,000 FE considering average solutions.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO
Ackley	0.8318565969	0.0	0.0	0.0
Griewank	0.0114923776	0.0139479208	0.0024653468	0.0131430472
Michalewicz	0.6046982133	0.0016370493	0.0016370493	0.0016370493
Non-continuous Rastrigin	11.0	0.3355457426	4.6666666667	2.4219316921
Rastrigin	49.4161921649	4.6431422664	6.3014073616	5.6381013235
Rosenbrock	0.7611433973	14.9739373973	16.6608073244	16.8734217044

Table D.14: PSO comparison for 1,500,000 FE considering average solutions.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO
Ackley	0.8318565969	0.0	0.0	0.0
Griewank	0.0114923776	0.0090172271	0.0196810216	0.0098507281
Hyper-ellipsoid	0.0	0.0	0.0000000002	0.0
Michalewicz	0.6046982133	0.0292622922	0.0141742391	0.0171950356
Non-continuous Rastrigin	11.0	5.3341696440	12.3333333333	3.3333333365
Parabola	0.0000000246	0.0000048125	0.0046133849	0.0000001343
Rastrigin	49.4161921649	14.2610764598	11.2761992885	15.5876851774
Rosenbrock	2.2940880387	20.1826294789	13.5157019334	14.4290959807
Sphere	0.0	0.0	0.0000000002	0.0

Table D.15: PSO comparison for 250,000 FE considering average solutions.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO
Ackley	0.8318565969	0.0	0.0	0.0
Griewank	0.0114923776	0.0155789208	0.0139462640	0.0156075740
Hyper-ellipsoid	0.0000004069	0.0127858108	0.0083253120	0.0029690743
Michalewicz	0.6063352625	0.1202174767	0.2031683446	0.0294789235
Non-continuous Rastrigin	11.3333333333	22.0000122989	26.3333333333	23.6685015395
Parabola	18.6038582709	27.1141543332	30.3665861017	9.9445068671
Rastrigin	49.4161921649	17.3599442294	22.5523935435	27.5271821093
Rosenbrock	15.9563162600	29.6712060238	31.7944580444	23.6019738315
Schaffer F6	0.0064772733	0.0	0.0	0.0
Sphere	0.0000055749	0.2673301964	2.4584737455	0.0724461950

Table D.16: PSO comparison for 3,000 FE considering average solutions.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO
Ackley	3.3231476775	3.6656615782	2.3471185663	3.7993752264
Alpine	0.0410857362	0.1058583180	0.0489896309	0.0571767603
Camel Back	0.0000000001	0.0	0.0	0.0
De Jong 5	0.0000000002	0.0	0.0	0.0
Drop Wave	0.0000003024	0.0000008637	0.0212515574	0.0000007274
Easom	0.0000687258	0.0000019163	0.0	0.3333333707
Gen... Penalized	23.4135697727	53.4320077974	8.1764346980	8299.8282069000
Griewank	1.7871371729	3.3811418140	1.1419446639	2.4779785042
Goldstein-Price	0.0000000006	0.0000000009	-0.0	0.0000000005
Hyper-ellipsoid	1888.4448456100	2656.9726578100	1491.2270736500	2029.5556637500
Michalewicz	1.4969071235	1.0209907626	1.7178090452	1.5065032175
Non... Rastrigin	103.6141028050	111.2885407850	87.6822596283	88.6883789954
Parabola	3418.9600152500	3915.7299502000	2773.0872997900	3125.3379437400
Rastrigin	164.5950651960	220.1112387230	121.5536371850	183.5392534760
Rosenbrock	710.1925387090	3773.9246038000	416.6967614490	2260.2515227600
Schaffer F6	0.0032388438	0.0097160726	0.0064773888	0.0097159102
Schwefel P2.22	6.2583974875	9.0250784778	4.8289577581	8.9679435527
Shubert	0.0000001906	0.0000007514	0.0000000044	0.0000009211
Sphere	20007.8587830000	26557.2125230000	16484.4946035000	26966.8690771000
Step	434.0	1152.3333333300	90.0	802.6666666670
Tripod	0.3339944284	1.3333832000	0.3333365596	0.0000749267

APPENDIX E

COMPARISON CONSIDERING HARDER PROBLEMS

Results for Ackley in the search space $[-30, 30]^{100}$, Alpine in the search space $[-10, 10]^{30}$, Generalized Penalized in the search space $[-50, 50]^{60}$, Griewank in the search space $[-300, 300]^{60}$, Hyper-ellipsoid in the search space $[-5.12, 5.12]^{100}$, Michalewicz in the search space $[0, \pi]^{30}$, Non-continuous Rastrigin in the search space $[-5.12, 5.12]^{60}$, Parabola in the search space $[-20, 20]^{200}$, Rastrigin in the search space $[-10, 10]^{60}$, Rosenbrock in the search space $[-10, 10]^{60}$, Schwefel P2.22 in the search space $[-10, 10]^{60}$, Sphere in the search space $[-100, 100]^{200}$, and Step in the search space $[-100, 100]^{60}$ are presented here. Michalewicz is the only problem for which changing the dimensionality caused a change in the optimum.

Table E.1: PSO vs. APSO for 15,000,000 FE, minimum solution, and increased dimensionality.

Test Problem	known PSO variants	proposed APSO variants
Michalewicz	-18.9908267260	-19.9701269255
Non-continuous Rastrigin	10.7183844953	0.0000000000
Rastrigin	24.8739713895	0.0000000000

Table E.2: PSO vs. APSO for 7,500,000 FE, minimum solution, and increased dimensionality.

Test Problem	known PSO variants	proposed APSO variants
Michalewicz	-19.3037120865	-19.3075176872
Non-continuous Rastrigin	19.0000000000	0.0000000000
Rastrigin	27.8588535986	6.9647133997

Table E.3: PSO vs. APSO for 250,000 FE, minimum solution, and increased dimensionality.

Test Problem	known PSO variants	proposed APSO variants
Ackley	0.0461864348	1.2371933991
Generalized Penalized	0.0000000000	0.0000000003
Hyper-ellipsoid	5.4276934402	13.6767854595
Michalewicz	-18.2583876155	-18.8608458011
Non-continuous Rastrigin	50.0000000000	50.0018868437
Parabola	31.3746026486	27.7319112256
Rastrigin	67.7114779078	62.7110070117
Rosenbrock	58.6327166169	52.3783792300
Schwefel P2.22	0.0000000376	0.0000055684
Sphere	176.4354493410	104.9260000520

Table E.4: PSO vs. APSO for 15,000,000 FE, average solution, and increased dimensionality.

Test Problem	known PSO variants	proposed APSO variants
Michalewicz	-18.8069840625	-18.7896250155
Non-continuous Rastrigin	22.0781662916	0.0000000000
Rastrigin	27.1955273941	1.6582650952
Rosenbrock	2.6577492362	0.0000000000

Table E.5: PSO vs. APSO for 7,500,000 FE, average solution, and increased dimensionality.

Test Problem	known PSO variants	proposed APSO variants
Griewank	0.0024653468	0.0090311322
Michalewicz	-18.7883124811	-18.7385036744
Non-continuous Rastrigin	42.3333333333	0.0000000000
Parabola	0.0000000000	0.0000093869
Rastrigin	38.1400753429	10.6128966090
Rosenbrock	2.6577492362	0.0000013250

Table E.6: PSO vs. APSO for 250,000 FE, average solution, and increased dimensionality.

Test Problem	known PSO variants	proposed APSO variants
Ackley	0.4290640146	1.6542867757
Generalized Penalized	0.0000397997	0.0000002838
Griewank	0.0000001342	0.0024653468
Hyper-ellipsoid	21.1553302540	28.2536405278
Michalewicz	-16.8733335825	-17.8030058809
Non-continuous Rastrigin	65.6666666667	64.0056749278
Parabola	83.0347416149	56.1199292424
Rastrigin	86.3060586187	95.3170764763
Rosenbrock	87.0409065313	86.5998983637
Schwefel P2.22	0.0000173135	0.0002465545
Sphere	285.6889640510	195.7078964200
Step	0.0000000000	0.3333333333