**SELF-ADAPTING CONTROL PARAMETERS IN PARTICLE SWARM
OPTIMIZATION**

by

MEWAEL DANIEL ISIET

B.Eng.Tech., Australian College of Kuwait, 2016

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Mechanical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

February 2019

© Mewael Daniel Isiet, 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, a thesis/dissertation entitled:

Self-Adapting Control Parameters in Particle Swarm Optimization

submitted by Mewael Daniel Isiet in partial fulfillment of the requirements

the degree of Master of Applied Science

in Mechanical Engineering

Examining Committee:

Dr. Mohamed S. Gadala

Supervisor

Dr. Sanja Miskovic

Supervisory Committee Member

Dr. Jasmin Jelovica

Supervisory Committee Member

Abstract

This study focuses on the development of a scheme for self-adapting the Particle Swarm Optimization (PSO) method to solve constrained optimization problems. PSO is a powerful nature-inspired metaheuristic optimization method. Compared to other methods, PSO has the ability to determine the optimal solution in fewer evaluations and in general performs in a more efficient and effective manner. However, researches show that the PSO method suffers from premature convergence and a dependence on the initial control settings. Due to these flaws, the application of PSO could lead to a failure in obtaining the global optimal solution.

An extensive parametric sensitivity analysis was conducted to understand the impact of the individual control parameters and their respective influence on the performance of PSO. Results of the sensitivity analysis revealed that PSO was most sensitive to the inertia weight, cognitive component and social component. Modifications were performed on the original PSO algorithm to adapt the control parameters with respect to the circumstances of the particles at a specific moment. The modified PSO variant is called the Unique Adaptive Particle Swarm Optimization (UAPSO). Unique control parameters were established for each particle through using a novel term known as the evolutionary state. In the developed approach, constraints were handled by forcing the particles to learn from their personal feasible solutions only. Therefore, in the proposed method, the constraint handling technique worked in accord with the adapting scheme to ensure that the particles were adapting to the environment by directing itself to the feasible regions. Furthermore, particles were reinitialized whenever they stagnated in the design space.

Verification of the performance of the proposed method was done by means of a comparative study with other well-known algorithms. The comparative study demonstrated that UAPSO proved to be effective and efficient in solving the considered problems and especially in terms of the speed of convergence. Furthermore, design of a three-bar truss was investigated through the application of UAPSO along with multiple variants of PSO. The numerical results showed the superiority of UAPSO compared to the other variants, its ability in avoiding premature convergence and its consistency and efficiency.

Lay Summary

Particle Swarm Optimization (PSO) is an optimization method that uses the concept of animal foraging in birds and fishes to determine the optimal properties of a design or system. Despite of all its advantages over other methods, PSO suffers from a condition that causes it to fail in obtaining the best possible solution to a given problem. In this thesis, an initial study was performed to determine the causes of failure within the algorithm and based on the findings, modifications were applied to PSO. A novel approach was proposed to allow the PSO algorithm to adapt itself based on the conditions of the problem-solving environment. By allowing the algorithm to direct itself automatically, the efficiency and effectiveness of the method increased. The performance of the proposed method demonstrated its potential in being applied to practical applications.

Preface

The research work presented in this thesis is an original and independent study by the author. All numerical experimental simulations and writings were done under the supervision of Dr. Mohamed S. Gadala in the Department of Mechanical Engineering, University of British Columbia. The developed MATLAB code is a novel modified version of the original Particle Swarm Optimization method. The following two manuscripts have been submitted for publication:

- “Sensitivity Analysis of Control Parameters in Particle Swarm Optimization” by M. Isiet and M. Gadala based on the results produced in Chapter 3.
- “Self-Adapting Control Parameters in Particle Swarm Optimization” by M. Isiet and M. Gadala based on the modified PSO method described in Chapter 4, the comparative study in Chapter 5 and the numerical results from Chapter 6.

Table of Contents

Abstract.....	iii
Lay Summary	iv
Preface.....	v
Table of Contents	vi
List of Tables	x
List of Figures.....	xi
List of Symbols, Greek Letters, Superscripts and Subscripts	xii
List of Abbreviations	xiv
Acknowledgments	xv
Dedication	xvi
Chapter 1: Introduction	1
1.1 Optimization	2
1.1.1 Types of Optimization Problems	2
1.2 Optimization Methods	3
1.2.1 Derivative – Based Search Methods	4
1.2.2 Zero – Order/Direct Search Methods.....	5
1.2.3 Derivative – Free Methods.....	6
1.2.4 Nature – Inspired Search Methods.....	6
1.3 Sample Applications of PSO	10
1.4 Research Objectives.....	11
1.5 Outline of the Thesis	11
Chapter 2: Particle Swarm Optimization	13

2.1	Control Parameters.....	15
2.1.1	Acceleration Coefficients.....	15
2.1.2	Velocity Clamping-Limit.....	16
2.1.3	Swarm Size	17
2.1.4	Maximum Number of Iterations	17
2.2	Inertia Weight	19
2.2.1	Variants of Inertia Weight	19
2.3	Constraint Handling Techniques.....	21
2.3.1	Proposed Constraint Handling Approach	22
2.4	Modified Variants of PSO	24
2.5	Summary	29
Chapter 3: Parameter Sensitivity Analysis.....		30
3.1	Parameter Sensitivity Analysis – Case Study	30
3.2	Numerical Experiment Setup.....	32
3.3	Numerical Experiment Results	33
3.3.1	Maximum Number of Iterations	33
3.3.2	Swarm Size	34
3.3.3	Inertia Weight	36
3.3.4	Acceleration coefficients	39
3.3.5	Clamping-Limit.....	41
3.4	Summary	42
Chapter 4: Proposed Method and Its Implementation		44
4.1	Adaptive Particle Swarm Optimization (APSO)	44

4.1.1	Motivation.....	45
4.2	Unique Adaptive Particle Swarm Optimization (UAPSO).....	46
4.2.1	UAPSO – Inertia Weight	47
4.2.2	UAPSO – Acceleration Coefficients	50
4.2.3	UAPSO – Reinitialization of Velocity.....	52
4.2.4	UAPSO – Repulsion Based Particle Velocity Update	52
4.3	Simulations on Benchmark Constrained Optimization Problems	55
4.3.1	Speed-Reducer	58
4.3.2	Pressure Vessel	60
4.3.3	Tension/Compression Spring.....	63
4.3.4	Welded Beam.....	65
4.4	Summary	68
Chapter 5: Application of the Proposed Method in Solving a Practical Structural		
Optimization Problem		70
5.1	Optimization Problem.....	71
5.2	Optimization Problem Setup.....	73
5.3	Numerical Results	74
5.3.1	Comparison of Convergence Performance	76
5.3.2	Performance of UAPSO with Varying Function Evaluations	78
5.4	Summary	80
Chapter 6: Conclusion.....		81
6.1	Contributions and Significance.....	82
6.2	Recommendations for Future Work.....	83

References	85
Appendices	99
Appendix A - MATLAB Code of the Speed-Reducer Design Problem.....	99
Appendix B - MATLAB Code of the Pressure-Vessel Design Problem.....	100
Appendix C - MATLAB Code of the Tension/Compression Spring Design Problem.....	101
Appendix D - MATLAB Code of the Welded Beam Design Problem	102
Appendix E - MATLAB Code of the Three-Bar Truss Design Problem	103
Appendix F - MATLAB Code of the Unique Adaptive Particle Swarm Optimization (UAPSO).....	104

List of Tables

Table 3-1 Mean Value and Standard Deviation (in brackets) with Varying Maximum Number of Iterations	33
Table 3-2 Mean Value and Standard Deviation (in brackets) with Varying Swarm Sizes.....	36
Table 3-3 Best, Worst, Mean and Standard Deviation (in brackets) with Varying Inertia Weight	37
Table 3-4 Best, Worst, Mean and Standard Deviation (in brackets) with Varying Acceleration Coefficients	39
Table 3-5 Best, Worst, Mean and Standard Deviation (in brackets) with Varying Clamping-Limit	42
Table 4-1 User Parameters for the Benchmark Design Problems	56
Table 4-2 Comparison of the Best Solutions for the Speed-Reducer Problem.....	59
Table 4-3 Comparison of the Statistical Performance for the Speed-Reducer Problem.....	60
Table 4-4 Comparison of the Best Solution for the Pressure-Vessel Problem	62
Table 4-5 Comparison of the Statistical Performance for the Pressure-Vessel Problem	62
Table 4-6 Comparison of the Best Solutions for the Tension/Compression Spring Design Problem	65
Table 4-7 Comparison of the Statistical Performance for the Tension/Compression Spring Design Problem	65
Table 4-8 Comparison of the Best Solutions for the Welded Beam Design Problem	68
Table 4-9 Comparison of the Statistical Performance for the Welded Beam Design Problem ...	68
Table 5-1 System Parameters of the Three-Bar Truss Design Problem	73
Table 5-2 Dynamic Range of the Considered PSO Variants	74
Table 5-3 Comparison of the Best Solutions for the Three-Bar Truss Design Problem	75
Table 5-4 Sum of Violated Constraints by the Considered PSO Variants and SQP	76
Table 5-5 Comparison between PSO-CIW with 20,000 FEs and UAPSO with 16,000 FEs	79

List of Figures

Figure 1-1 Overview of the modifications introduced in to the PSO method	12
Figure 2-1 Particle Velocity and Position Update in a Two-Dimensional Search Space	14
Figure 2-2 Effect of Clamping-Limit in Updating the Position of Particles.....	16
Figure 2-3 Particle Swarm Optimization Flowchart	18
Figure 2-4 Schematic Diagram of the Constraint Handling Approach - 1st Condition.....	23
Figure 2-5 Schematic Diagram of the Constraint Handling Approach - 2nd Condition	24
Figure 3-1 Schematic Representation of the Speed-Reducer Design Problem.....	30
Figure 3-2 Performance (Best and Worst Value) of PSO with Varying Maximum Number of Iterations	34
Figure 3-3 Performance (Best and Worst Value) of PSO with Varying Swarm Sizes.....	35
Figure 3-4 Dynamic Behavior of the Considered Inertia Weight Variants	36
Figure 3-5 Comparison of the Momentum-Less PSO and the Chaotic Inertia Weight PSO in Terms of Convergence	38
Figure 3-6 Comparison of the Convergence Curves of the Conventional Acceleration Coefficient and TVAC	40
Figure 4-1 Schematic description of low Evolutionary State when $f(x_f^{(i,k)}) \neq f(x_p^{(i,k)})$ or/and $f(x_f^{(i,k)}) \neq f(x_g^{(k)})$	48
Figure 4-2 Schematic description of high Evolutionary State when $f(x_f^{(i,k)}) \neq f(x_p^{(i,k)})$ or/and $f(x_f^{(i,k)}) \neq f(x_g^{(k)})$	48
Figure 4-3 Dynamic Behavior of the Inertia Weight in UAPSO.....	49
Figure 4-4 Dynamic behavior of Acceleration Coefficients – UAPSO.....	51
Figure 4-5 Pseudocode – UAPSO.....	54
Figure 4-6 Schematic Representation of the Pressure-Vessel Design Problem	60
Figure 4-7 Schematic Representation of the Tension/Compression Spring Design Problem	63
Figure 4-8 Schematic Representation of the Welded-Beam Design Problem [105]	65
Figure 5-1 Schematic Diagram of the Three-Bar Truss Design Problem.....	71
Figure 5-2 Statistical Performance of UAPSO and the Considered PSO Variants in the Three-Bar Truss Problem	75
Figure 5-3 Convergence Curves of UAPSO and the Considered Variants for the Three-Bar Truss Design Problem.....	77
Figure 5-4 Statistical Performance of UAPSO with Varying Function Evaluations.....	78
Figure 5-5 Comparison of the Mean Function Value and Distribution of Results Produced by UAPSO (with 50% less FEs) with the Considered PSO Variants.	79

List of Symbols, Greek Letters, Superscripts and Subscripts

List of Symbols

a_1, a_2, a_3	Cross-sectional area
$b(x)$	Equality constraint
c	Nonlinear factor
$c(x)$	Inequality constraint
c_1, c_2, c_3	Acceleration coefficient
E	Young's Modulus, <i>GPa</i>
$f(x)$	Objective function
$F(x)$	Penalty function
g	Gravity, m/s^2
h	Violated constraint
K	Stiffness matrix
k	Current iteration
L	Length, m
M	Mass matrix
Max_k	Maximum iteration
N	Swam size
P	Load, kN
$rand_1, rand_2$	Random number
S_a	Allowable strength, <i>MPa</i>
u	Horizontal displacement, mm
v	Vertical displacement, mm
$v^{(i, k)}$	Current particle velocity
$v^{(i, k+1)}$	Updated particle velocity
V_{max}	Clamping Limit
w	Inertia weight
x	Design variables
x_1, x_2, x_3	Horizontal coordinate
$x^{(i, k)}$	Current particle solution
$x^{(i, k+1)}$	Updated particle solution
$x_f^{(i,k)}$	Latest feasible solution of a particle
$x_G^{(k)}$	Global best solution of the swarm
x_L	Lower limit of design variable
x_N	Number of design variables
$x_P^{(i,k)}$	Personal best solution of a particle
x_U	Upper limit of design variable
y	Constraint value
z	Chaotic term

List of Greek Letters

β_1, β_2	Penalty factor
∇	Gradient
δ	Maximum relative displacement, <i>mm</i>
θ	Angle of applied load, $^\circ$
ρ	Material density, <i>kg/m³</i>
ω	Natural frequency, <i>Hz</i>

List of Superscripts & Subscripts

<i>i</i>	Particle number
<i>k</i>	Iteration
<i>max</i>	Maximum value
<i>min</i>	Minimum value

List of Abbreviations

ACO	Ant Colony Optimization
APSO	Adaptive Particle Swarm Optimization
CIW	Chaotic Inertia Weight
GA	Genetic Algorithm
HPSO – TVAC	Hierarchical Particle Swarm Optimizer with Time-Varying Acceleration Coefficients
LDCL	Linearly Decreasing Clamping-Limit
LDIW	Linear Decreasing Inertia Weight
NLIW	Nonlinear Inertia Weight
PSO	Particle Swarm Optimization
RIW	Random Inertia Weight
RPSO	Repulsive Particle Swarm Optimization
UAPSO	Unique Adaptive Particle Swarm Optimization

Acknowledgments

Firstly, I want to give thanks to the beneficent and merciful God for His guiding presence throughout my studies and for endowing me with the strength to accomplish my research.

I want to express my gratitude to my supervisor, Dr. Mohamed S. Gadala, for taking me on and providing me the chance to work with him in UBC. Without his support and guidance, I would not have been able to complete this research work.

Major thanks go to Dr. Mohsen Seraj for his kindness and advices throughout my time in UBC. I want to express my appreciation to all current and former members of X221– thanks for the fruitful conversations, the laughs and the friendly environment.

Lastly the love, prayers and support from my family and friends were my driving force, big thanks go to you.

Dedication

To my mother, who had to go through many hardships and difficulties to bring me up.

Chapter 1: Introduction

Determining the optimal parameter(s) of any service, product or project leads to unlocking its full potential. Maximizing profit and minimizing waste are two of the most important factors considered when designing any product. In engineering, products are manufactured regardless if they are the optimal design or not, since it would be costly in terms of time and money, as it would require human and material resources to optimize. For a specific product, multiple designs could be produced, and tested, to serve the same purpose, however there exists in theory other designs or a design that could perform more effectively and efficiently but due to time and money constraints, a design is selected based on initial tests.

The design problem could be formulated into an optimization problem where the desired properties are optimized, while simultaneously satisfying performance related and manufacturing specifications and constraints. If a system or design is defined as a process that consists of inputs that perform certain functions to produce outputs, then by producing a mathematical representation of the design and introducing an evaluation criterion, an optimization problem is created. Therefore, an optimization problem consists of three components, a property to be minimized or maximized, restriction(s) or constraint(s) that needs to be satisfied and parameter(s) that can be adjusted. The parameter, either explicitly or implicitly, has an effect on both the property being optimized and the restriction that is being fulfilled. Many optimization methods have been developed and applied to produce better designs. Due to the advancement in computational power and the increasing need to solve optimization problems, researchers have developed various optimization methods to meet the demands to solve complex and multidimensional problems. This research focuses on the metaheuristic nature – inspired optimization method, the Particle Swarm Optimization (PSO) algorithm. PSO is a stochastic optimization method that takes its inspiration

from the social behavior of animals looking for food such as a flock of birds or a school of fish [1].

1.1 Optimization

Generally speaking, the standard mathematical representation of an optimization problem is stated as:

$$\text{minimize} \quad f_i(\mathbf{x}) = f_i(x_1, x_2, \dots, x_n) \quad \text{for } i = 1 \text{ to } m \quad 1.1$$

$$\text{subject to} \quad b_j(\mathbf{x}) = b_j(x_1, x_2, \dots, x_n) = 0 \quad \text{for } j = 1 \text{ to } n \quad 1.2$$

$$c_k(\mathbf{x}) = c_k(x_1, x_2, \dots, x_n) \leq 0 \quad \text{for } k = 1 \text{ to } o \quad 1.3$$

$$\mathbf{x}_L \leq (x_1, x_2, \dots, x_n) \leq \mathbf{x}_U \quad 1.4$$

where $f_i(\mathbf{x})$ is the objective function, in cases where multiple objective functions are present in the problem $i > 1$ and $(x_1, x_2, \dots, x_n)$ represents the design variables. $b_j(\mathbf{x})$ and $c_k(\mathbf{x})$ are the referred to as equality and inequality constraints respectively. Finally, the design variables are bounded by a lower limit, $\mathbf{x}_L$, and an upper limit, $\mathbf{x}_U$.

1.1.1 Types of Optimization Problems

There exist many types of optimization problems depending on the nature of the mathematical formulation, the type of allowable design variable, behaviour of the functions etc. Some definitions of the different optimization problems are given below:

- **Integer/Discrete and Continuous Programming Problems:** In integer programming problems, the design variables have to be an integer value. Problems where some variables are required to be integers are termed as mixed integer programming problems. Whereas problems where any value can be selected within a predefined set are referred as continuous programming problems [2].

- **Linear & Nonlinear Programming Problems:** Optimization problems where the objective function(s) and constraint(s) are all linear are known as linear programming problems. Nonlinear programming problems are problems where nonlinearity exists in any of the of the optimization functions [3].
- **Dynamic & Static Optimization Problems:** In dynamic optimization problems, the objective function(s) and constraint(s) are solved and optimized for every time t . In problems where the optimization function(s) are static throughout all iterations, this is known as a static problem [4].
- **Constrained & Unconstrained Problems:** Problems where the objective function(s) is solved while fulfilling some restriction(s) applied to the design variables are called constrained optimization problems. Whereas a problem where the objective function(s) is solved without any restriction(s) is known as unconstrained problem.
- **Single Objective & Multi-Objective Function Problems:** In cases where the optimization problem contains one objective function, this problem is known as a single objective function problem. While in cases where multiple objective functions are solved simultaneously, this type of problem is referred to as multiobjective function optimization problem.

1.2 Optimization Methods

Depending on factors such as the complexity of the problem, availability of time, computational power, knowledge of behavior of the functions etc., an optimization method is selected [2]. The following subsections describe the various classes of optimization methods.

1.2.1 Derivative – Based Search Methods

As the name suggests, this class of methods requires derivative information to solve the optimization problem. These methods perform an iterative calculation by initially starting with an estimate and attempts to improve the design until certain mathematical conditions are fulfilled. The earliest derivative based method developed is the steepest descent method [2]. The goal of this method is to find the search direction, $\mathbf{d}_k$ at iteration k that would reduce the objective function in the fastest way. The design variable update equation is described below [2]:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \quad 1.5$$

where α_k is the step size that ensures that at every iteration the objective function reduces along the search direction. The steepest descent method utilizes first order or gradient information of the objective function to solve for $\mathbf{d}_k$. A method that utilizes the second order information or Hessian of the functions is the Newton's Method. The Hessian of the objective function is used to determine $\mathbf{d}_k$. Both the steepest descent and the Newton's Method have been formulated to solve unconstrained problems only. Therefore, for solving constrained problems the Sequential Quadratic Programming (SQP) method could be used [2]. The SQP method formulates a quadratic subproblem through the quadratic approximation of the Lagrange function. This is given by [5]:

$$\text{minimize} \quad \nabla f(\mathbf{x}_k)^T \mathbf{d} + 0.5 \mathbf{d}^T \mathbf{H}_k \mathbf{d} \quad 1.6$$

$$\text{subject to} \quad \nabla b_i(\mathbf{x}_k)^T \mathbf{d} + b_i(\mathbf{x}_k) = 0 \quad 1.7$$

$$\nabla c_i(\mathbf{x}_k)^T \mathbf{d} + c_i(\mathbf{x}_k) \leq 0 \quad 1.8$$

where $\mathbf{H}_k$ is the approximated Hessian matrix, ∇b_i is the gradient of the equality constraint, ∇c_i is the gradient of the in-equality constraint and ∇f is the gradient of the objective function [6].

1.2.2 Zero – Order/Direct Search Methods

These methods do not require any first or second derivative information, which is the reason why they are known as zero-order method. Only information from the objective function(s) and constraint(s) are used by these methods [7]. The key feature of this class of methods is that they do not utilize any qualitative information related to the function values and that no approximation of the optimization function(s) is made [8]. A common direct search method is the pattern search method, the Hooke-Jeeves method [9]. The Hooke-Jeeves method performs a series of exploratory and pattern moves to produce an improved design. Here also the iterative calculation is initiated with an initial design. In the exploratory phase, a single dimensional coordinate, i.e. a single design variable, is adjusted with respect to a predetermined step size. If the function did not increase, the coordinate value is stored and if the function increased, the value is rejected and the step size is shortened by a factor and then depending on its performance, the value is either stored or rejected. The exploratory iterative equation is described below [9]:

$$x_{k+1}^i = x_k^i + a_k^i \quad 1.9$$

where x_k^i is an individual design variable at iteration k and a_k^i refers to the step size. Once all i coordinates have been investigated and successfully updated, the location of the design point is termed as the base point. The difference between the present and previous base point constitutes the search direction taken in the pattern search phase. The pattern search equation is described by [9]:

$$\tilde{\mathbf{x}} = \mathbf{x}_k + \mathbf{d}_k; \quad \mathbf{d}_k = \mathbf{x}_k - \mathbf{x}_{k-1} \quad 1.10$$

where $\tilde{\mathbf{x}}$ is the temporary base point and $\mathbf{d}_k$ is the search direction. $\tilde{\mathbf{x}}$ is accepted as the new base point only if $f(\tilde{\mathbf{x}}) \leq f(\mathbf{x}_k)$, if not the exploratory phase is restarted. The search continues until it

reaches a point where no further improvement is made [10]. Another popular direct search method is the Nelder – Mead simplex method [11].

1.2.3 Derivative – Free Methods

The derivative – free methods, just like the direct search methods, do not require any first or second order information from the optimization function [8]. However, the derivative – free methods approximate the optimization functions using a method called the response surface methods (RSM). RSM is a technique, which is used to generate an approximate relationship between the design variables and the output parameter [12]. The original system is evaluated at several design points and after which a correlation is made using linear, quadratic or cubic functions. In order to produce a response surface, design points need to be selected. The selection of design points can be performed either randomly or by utilizing methods that attempt to cover the entire design space. Using the method of Design of Experiments (DOE), sampling points can be determined for generation of the response surface model. Some examples of DOE are the Central Composite Design [12] and the Box – Behnken [13]. Once the approximated model has been generated, any optimization method may be applied onto it. The response surface method, compared to the original system, reduces the computational demands; however, depending on the quality of the approximation, the result accuracy would be affected [14].

1.2.4 Nature – Inspired Search Methods

This class of methods can be considered as a subclass of the zero – order search methods. In this class, stochastic behavior is imbedded in the algorithm leading to a certain degree of randomness in the problem solving. This randomness is desirable since it allows the algorithm to escape a local result and increases the possibility of locating the global solution [15]. Due to the advancement in computational power, this class of methods is becoming widely used to solve

optimization problems. These methods take their inspiration from naturally occurring phenomena such as natural selection and swarm intelligence [15]. Furthermore, these methods are general which means that they could be used to solve any type of optimization problem. The following subsections will cover some techniques within the nature – inspired class namely the Genetic Algorithm, Ant Colony Optimization and Particle Swarm Optimization.

1.2.4.1 Genetic Algorithm

A popular stochastic method, belonging to the nature-based search methods, is the Genetic Algorithm (GA). GA solves for the global optimum by using the concept of natural selection [16]. In GA, design points are initialized randomly with respect to the allowable values for each design variable and the fitness of each randomly assigned design point is evaluated through the objective function. Two common terms used in GA are chromosome and gene [15]. Chromosome refers to the design point (vector containing the value of all design variables) and gene refers to the scalar quantity of an individual design variable. The GA algorithm usually represents the chromosome in the form of a binary string. This algorithm contains three key steps: Selection, Crossover and Mutation. After the evaluation of each chromosome, the ones with the best potential are selected for reproduction which comprises the selection step [16]. In the crossover step, the algorithm merges two chromosomes (parents) to form a new design point (child). Mutation is the following step in the GA algorithm where the objective is to reintroduce any lost genetic information in the preceding steps or to introduce any new information [17]. In this step, the binary value of the gene undergoes modifications (from 0 to 1 or vice versa). Modification to the chromosome is necessary since it helps the chromosome in getting out of a local solution and continuing the search for the global solution [15].

1.2.4.2 Ant Colony Optimization

When searching for food, ants leave a trail of pheromone behind to help lead other members of their colony to a source of food. An individual member will initially be randomly looking for food and upon detection of the pheromone trail, it will deduce whether to follow it or not. By following the same path, an ant secretes its own pheromone onto the trail thus increasing the likelihood of other ants following the same path [18]. Furthermore, it has been experimentally shown that this behavior (pheromone trail) gives rise to the shortest path between the colony and food [4]. The Ant Colony Optimization (ACO) is a swarm intelligence based method emulating the foraging behavior of ants [19]. Simply put, in the ACO algorithm the goal is to find the shortest path between two points, the ant's initial position and the food source (optimal solution). All nodes are connected by a series of discrete points forming links. These links represent the trail that an ant would follow through when foraging. All links are initialized with a pheromone value and the virtual ants moves to the next node through a probabilistic method until they reach their final destination. Upon reaching the end point, evaporation of pheromone occurs leading to a reduction in the pheromone value across all links. Following this, the ants make their way back to their initial position while reinforcing their chosen path by updating the pheromone value [18]. As could be understood from the above description, ACO was developed to solve discrete problems but studies [20, 21] have been performed to expand the algorithm to solve for continuous optimization problems.

1.2.4.3 Particle Swarm Optimization (PSO)

Also belonging to the swarm intelligence based optimization methods, the PSO algorithm is a stochastic population based metaheuristic method that takes its inspiration from the social behavior of animals looking for food such as flocking with birds and schooling with fish [1]. In

nature, a member of a swarm depends on its own intelligence and that of the swarm when searching for food. If any individual discovers that another member of the swarm is following a better path to food, it alters its movement to follow that path no matter where it was originally located. In PSO, particles fly through the hyperdimensional design space at a random velocity initially. The position of these particles change based on their personal history and the experience of the swarm as a whole. Each particle memorizes its best solution throughout every iteration and the best (global) solution across all particles. Therefore, memory is a key component in this algorithm. Here a swarm refers to a group of particles, while a particle is an individual member of a swarm, i.e. a design point. The balance between the local and global information is vital because it allows particles to discover new locations and be attracted to previously successful regions in the design space [22]. There exists some similar concepts between GA and PSO [23], in PSO the concept of crossover is somewhat present in the form of particle attraction to its best solution and that of the swarm and as for mutation, the particles velocity is modified using a vector with a direction lying in-between its personal best and the global best [24]. However, the selection concept in GA is not present in PSO. PSO compared to the other methods has the following advantages [25]:

- PSO manages to obtain the same quality of results compared to other methods at a lower computational time (requiring less iterations).
- Comparing PSO to other methods, it is demonstrated (in literature) that PSO manages to be effective in obtaining better solutions, efficient in the accuracy of results and robust in escaping from local solutions.
- The PSO method does not require any knowledge on the design domain to solve the problem and is thus intuitive. The method can be coded in a few lines and does not contain any complicated mathematical functions or expressions.

1.3 Sample Applications of PSO

- A comparison between PSO and GA in designing a Flexible AC Transmission System (FACTS)-based controller demonstrated that even though both methods managed to optimize the controller, PSO performed better and required fewer iterations than GA to solve the problem [26].
- Simulation results obtained for the optimization of the design of scannable circular antenna arrays by GA, PSO and the differential method displayed that under equal computational time the differential method and PSO managed to perform better than GA. Furthermore, comparing the mean objective function after 50 trials it was observed that PSO performed the best [27].
- A study was performed on a feed forward neural network learning a nonlinear function using PSO and backpropagation. It was demonstrated that PSO required fewer number of iterations to solve the problem compared to backpropagation and it was concluded that PSO is the better option for applications that need quick learning algorithms [28].
- A method was developed to predict crystal structure through PSO. In order to validate the performance of the proposed method, a comparison was made with respect to the same solution obtained from GA. The comparison demonstrated that the proposed PSO methodology outperformed GA in computational time [29].
- Minimizing cost in executing applications within Cloud computing environments was the objective of a study by using PSO and comparing its performance with an existing method known as the Best Resource Selection. It was found that PSO managed to reduce cost by threefold compared to Best Resource Selection [30].

1.4 Research Objectives

The main objective of the thesis is to develop a self-automated PSO algorithm for solving constrained optimization problems. Failure in determining the optimal solution is linked to poor tuning of control settings [31] leading to premature convergence which refers to a state where particles are unable to escape from a local solution and thus converge prematurely before the discovery of the global solution [32]. Thus, it is important to know the degree of sensitivity of each individual control parameter in context of the algorithm's performance. The parametric sensitivity of PSO is investigated, in this thesis, by means of a simple approach where individual parameters are varied independently. Statistical data is collected and the degree of sensitivity of a parameter is determined. Novel modifications are, then, introduced into the PSO algorithm to make it adaptive to the environment of the problem and enables the algorithm to update itself accordingly.

The following lists the objectives of this research work:

- Explore the multiple control parameters in PSO and study its sensitivity by monitoring its impact on the PSO algorithm's performance.
- Develop modification(s) to the original algorithm and validate the performance of the modified method.
- Investigate the proposed method's ability in solving real-world structural optimization design problems.

1.5 Outline of the Thesis

The organization of this thesis comprises of six chapters. In the present chapter, a brief introduction to optimization and the different types of optimization problems were presented. Chapter 2 introduces the PSO algorithm, its control parameters, the proposed constraint handling approach and some variants of the original PSO algorithm. Chapter 3 investigates the sensitivity

of PSO's control parameters with respect to its influence on the performance of the algorithm. Chapter 4 describes the proposed adaptive PSO method, its application to some benchmark examples and validation of performance through comparison with literature. Here, the scheme of adapting the control parameters through a feedback system is introduced. Figure 1-1 gives an overview of the modifications applied in the proposed method. In Chapter 5, the proposed method is applied to a practical structural optimization problem and its performance is compared with some variants of PSO. Finally, Chapter 6 presents the conclusion and contributions of the thesis. Furthermore, in this chapter suggestions are also made for future work in this line of research.

Figure 1-1 Overview of the modifications introduced in to the PSO method

Chapter 2: Particle Swarm Optimization

In PSO [1], each particle represents a design point in the hyperspace and its current location is denoted by $\mathbf{x}^{(i,k)}$ where i is the individual particle and k is the iteration number. Each particle keeps track of its current position and its best solution so far denoted by $\mathbf{x}_p^{(i,k)}$. Another value that is tracked by the algorithm is the best global solution denoted by $\mathbf{x}_G^{(k)}$. The particle velocity $\mathbf{v}^{(i,k)}$ refers to the change in particle location at each iteration k [33]. In short, the PSO algorithm attempts to accelerate particles by updating its velocity towards its own personal best and the global best solution. A particle's velocity and position, $\mathbf{v}^{(i,k)}$ and $\mathbf{x}^{(i,k)}$ respectively, are iteratively updated by the following two equations [1]:

$$\mathbf{v}^{(i,k+1)} = \mathbf{v}^{(i,k)} + c_1 \mathbf{rand}_1^{(k)} (\mathbf{x}_p^{(i,k)} - \mathbf{x}^{(i,k)}) + c_2 \mathbf{rand}_2^{(k)} (\mathbf{x}_G^{(k)} - \mathbf{x}^{(i,k)}); i = 1 \text{ to } N \quad 2.1$$

$$\mathbf{x}^{(i,k+1)} = \mathbf{x}^{(i,k)} + \mathbf{v}^{(i,k+1)}; i = 1 \text{ to } N \quad 2.2$$

where c_1 and c_2 are the cognitive and social parameter and are collectively known as the acceleration coefficients. $\mathbf{rand}_1^{(k)}$ and $\mathbf{rand}_2^{(k)}$ are vectors containing random numbers generated at every k iteration ($0 \leq rand \leq 1$) and N refers to the swarm size. In the particle velocity-update equation, three vector components exist – the momentum vector, the cognitive or personal component and the social or global component. Figure 2-1 illustrates how the three velocity vector components influences a particle towards the search for the global best solution. The above two equations represent the original PSO algorithm. Once the particles' new velocity and position has been calculated, $\mathbf{x}_p^{(i,k)}$ and $\mathbf{x}_G^{(k)}$ are updated iteratively with the following equations respectively [1]:

$$\mathbf{x}_p^{(i,k)} = \begin{cases} \mathbf{x}^{(i,k)}, & \text{if } f(\mathbf{x}^{(i,k)}) < f(\mathbf{x}_p^{(i,k)}) \\ \mathbf{x}_p^{(i,k)}, & \text{if } f(\mathbf{x}^{(i,k)}) \geq f(\mathbf{x}_p^{(i,k)}) \end{cases} \quad 2.3$$

$$\mathbf{x}_G^{(k)} = \begin{cases} \mathbf{x}^{(i,k)}, & \text{if } f(\mathbf{x}^{(i,k)}) < f(\mathbf{x}_G^{(k)}) \\ \mathbf{x}_G^{(k)}, & \text{if } f(\mathbf{x}^{(i,k)}) \geq f(\mathbf{x}_G^{(k)}) \end{cases} \quad 2.4$$

Both the particle's position and velocity are bounded according to a predefined limit. The dynamic range of a particle's position is defined by the design variable limit, whereas the particle's velocity range is defined by the velocity clamping-limit. In terms of initialization, the particles' positions are randomly distributed across the hyperspace, while their velocities are set as zero [34]. The efficiency of the algorithm is highly influenced by the initial diversity of the particles since the more diverse the particles are, the higher the chances of them locating the global solution are [22]. In the original PSO version [1], particles tend to be attracted to the global leader and thus it is defined by a global topology. Topology is a term used to describe how particles learn from each other and the extent of its neighborhood [35]. The interaction of a particle with its neighbor directly affects its search information and space [36]. To date there exists multiple types of topologies such as the global topology [1], local best topology [33], von Neumann topology [37] and the

Figure 2-1 Particle Velocity and Position Update in a Two-Dimensional Search Space

pyramid topology [37]. Studies comparing the different topology concluded that each type has its own advantages and disadvantages [36, 38]. Equations 2.1 ~ 2.4 are updated at every iteration until a predefined termination criterion is met. Typically, the algorithm terminates once an acceptable global solution is obtained or a predefined maximum number of function evaluations or iterations is met [39].

2.1 Control Parameters

The original PSO version described in the previous section has certain parameters, known as the control parameters, which have an impact on the overall search capabilities of the algorithm. These control parameters, namely the acceleration coefficients, velocity clamping-limit, swarm size and maximum number of iterations, are described in the following subsections.

2.1.1 Acceleration Coefficients

The terms c_1 and c_2 when changed could lead to particles flying to undesirable regions, therefore the acceleration coefficients have both been set as 2 since the conception of the PSO method [1]. However, this could be harmful to the overall search for the optimal solution since these parameters are key in controlling the effectiveness and efficiency of the algorithm [40]. Therefore, a time varying acceleration coefficient was developed to guide the particles to a global search ($c_1 > c_2$) in the initial stages and then gradually direct them to a localized search ($c_2 > c_1$) in the final iterations [41]. The acceleration coefficients time varying forms are presented below [41]:

$$c_1^k = \left(\frac{(c_{1max} - c_{1min}) \times (Max_k - k)}{Max_k} \right) + c_{1min} \quad 2.5$$

$$c_2^k = c_{2max} - \left(\frac{(c_{2max} - c_{2min}) \times (Max_k - it)}{Max_k} \right) \quad 2.6$$

where c_1^k and c_2^k are the cognitive and social parameters respectively at iteration k , c_{1max} and c_{2max} are the maximum cognitive and social parameter respectively, c_{1min} and c_{2min} are the minimum cognitive and social parameter respectively, k is the current iteration and Max_k is the maximum iteration.

2.1.2 Velocity Clamping-Limit

In the order to clamp the maximum step size change, the particles' velocities are bounded by $[-V_{max} V_{max}]$ [42]. The value of the maximum clamping-limit controls the coarseness of the search by restricting the step size (velocity) changes [43]. When a large V_{max} is set, global exploration is promoted, however a very large value could lead to particles overflying the global optimum leading to premature convergence. On the other hand, a smaller V_{max} facilitates local search but a very small value would lead to particles requiring more iterations to solve the problem

Figure 2-2 Effect of Clamping-Limit in Updating the Position of Particles

[44, 45]. Conventionally the value of V_{max} is set between (0.1~1.0) times $\mathbf{x}_{max}$ (where $\mathbf{x}_{max}$ is a vector containing the maximum value of the design variables) [46]. The effect of particle velocity clamping-limit for a two-dimensional design space is illustrated in Figure 2-2. In this figure, $\mathbf{x}^{(i,k+1)'}$ represents the position of particle i without the effect of the clamping-limit. The position $\mathbf{x}^{(i,k+1)}$ is the result of the clamping-limit on both dimensions, x_1 and x_2 . The velocity $\mathbf{v}^{(i, k+1)'}$ is the velocity of particle i without the clamping-limit and $\mathbf{v}^{(i, k+1)}$ with clamping-limit.

2.1.3 Swarm Size

Swarm size, N , plays an important role in the performance of the algorithm. This property refers to the number of particles in the hyperspace – it explicitly affects the diversity in the swarm. When a large swarm size is selected, the particles cover more space in the search area but at the expense of computational efforts. According to Arora [2], anywhere between $5x_N \sim 10x_N$ (where x_N is the number of design variables) is a good estimate, however in some other researches anywhere between 10 ~ 30 particles [47, 48, 49] proved to be successful in solving common optimization benchmark problems.

2.1.4 Maximum Number of Iterations

Maximum number of iterations, Max_k , refers to the number of iterations the PSO algorithm goes through until it terminates. If the maximum number of iterations is set as the termination criteria, then the predefined value has a direct impact on the likelihood of the algorithm discovering the global solution. Therefore, when a smaller value is selected, the probability of PSO obtaining the global solution is lower than if a larger value was selected. However, the downside of a large maximum number of iterations is the increased computational cost [44]. Therefore, the phenomena of premature convergence could occur due to the poor selection of a maximum number of

iterations. A visual description of the original PSO algorithm is presented in the form of a flow chart in Figure 2-3.

Figure 2-3 Particle Swarm Optimization Flowchart

2.2 Inertia Weight

A research [50] studying the original PSO version noted that in the absence of the momentum component (first right-hand term in Equation 2.1), the particles' velocity would only be influenced towards their personal and global best solutions. Therefore, particles located at the global best solution would be inactive since in the absence of momentum, they would have zero acceleration. Only after the discovery of a new global best solution, those particles would be active again. Thus, it was concluded that the momentum-less version promotes a refined or localized search. The research also studied the performance of PSO purely based on the momentum component. It was demonstrated that the momentum-only version restricts the particles into being at a constant state of flying. Consequently, it was concluded that the momentum-only version encourages a global or explorative search. Based on the conclusions from the two researched versions, a new parameter was proposed to help control the local and global search known as the inertia weight [50]. This new parameter was incorporated into Equation 2.1 and is displayed below [50]:

$$\mathbf{v}^{(i, k+1)} = w\mathbf{v}^{(i, k)} + c_1\mathbf{rand}_1^{(k)} \left(\mathbf{x}_p^{(i,k)} - \mathbf{x}^{(i,k)} \right) + c_2\mathbf{rand}_2^{(k)} \left(\mathbf{x}_G^{(k)} - \mathbf{x}^{(i,k)} \right); \quad i = 1 \text{ to } N \quad 2.7$$

where w is an inertia damping term that impacts the effect of the previous velocity to the current velocity. A large value enhances the particle's global search and a small value increases particle's local search.

2.2.1 Variants of Inertia Weight

Early on researchers, while comparing the performance of multiple constant inertia weights with a linearly decreasing form, demonstrated a significant improvement in the performance of PSO when the inertia weight was set between a range of 0.9 to 1.2 [50]. In a subsequent study on the linearly decreasing inertia weight [51], it was reported that varying the inertia weight between 0.9

to 0.4 resulted in an improved performance. The linearly decreasing inertia weight equation is described below [44]:

$$w^{(k)} = \left(\frac{(w_{max} - w_{min}) \times (Max_k - k)}{Max_k} \right) + w_{min} \quad 2.8$$

where w_{min} is the minimum inertia weight value, w_{max} is the maximum inertia weight, k is the current iteration and Max_k is the maximum iteration. $w^{(k)}$ decreases from w_{max} to w_{min} with respect to k and Max_k . Naturally, different dynamic strategies could be applied to the inertia weight. For example, the adaption of the inertia weight by a random distribution $\left(w^k = 0.5 + \frac{rand}{2} \right)$ was reported to have improved the robustness of the PSO algorithm [52]. Another effective strategy was the use of a fuzzy system to adapt the inertia weight, which proved to be effective in solving dynamic optimization problems [53]. Some researchers utilized previously developed versions of the inertia weight to enhance its influence in the particles' search capability. By introducing an exponent component into Equation 2.8, the nonlinear inertia weight [54] was devised with the following equation:

$$w^{(k)} = \left(\frac{(w_{max} - w_{min}) \times (Max_k - k)^c}{(Max_k)^c} \right) + w_{min} \quad 2.9$$

where c is a exponential factor introduced into the linearly decreasing inertia weight equation. It was reported that when $c = 0.9 \sim 1.3$ the inertia weight managed to efficiently navigate the particles between a globalized search in the early iterations to a more refined search in the latter iterations [54]. In another adaption, a chaotic term was introduced into the linearly decreasing inertia weight to create the chaotic inertia weight with the below form [55]:

$$w^{(k)} = \left(\frac{(w_{max} - w_{min}) \times (Max_k - k)}{(Max_k)} + w_{min} \right) \times z^k \quad 2.10$$

$$z^k = \alpha \times z^{k-1} \times (1 - z^{k-1})$$

where z is the chaotic term and when $3.75 \leq \alpha \leq 4$ the inertia weight behaves in a chaotic manner. Moreover, when $\alpha = 4$, the chaotic term covers an interval of $[0, 1]$ provided that $z^0 \in [0, 1]$ and $z^0 \notin [0, 0.25, 0.5, 0.75, 1.0]$. This strategy promotes more randomness and thus increases the swarm's diversity [56]. Furthermore, empirical results demonstrated that the chaotic inertia weight managed to improve PSO's convergence accuracy, convergence speed and exploration search ability [55].

2.3 Constraint Handling Techniques

Many real-life engineering design problems are formulated as constrained optimization problems. Presence of constraints have a significant effect on the performance of an optimization algorithm. The original form of any nature inspired optimization method lacks a system for handling constraints [57]. Constraint handling techniques are required to provide necessary information pertaining to the feasibility of a specific solution [58]. The general taxonomy of constraint handling techniques is classified as [57]:

- **Penalty Functions:** The constrained optimization problem is converted into an unconstrained problem by incorporating a penalty parameter.
- **Decoders:** In this technique, the feasible region of a problem is mapped into a simpler space to help enable better performance by the optimizer. Each solution in the decoded space corresponds to a solution in the design space.
- **Special Operators:** A special operator is considered as a method of either preserving a solution's feasibility or moving within a specific area of interest within the design space.

- **Separation of Objective Function and Constraints:** A two-step method where the aim of the first phase is to determine feasible solutions (regardless of the objective function value) and the second phase goal is to optimize the objective function.

2.3.1 Proposed Constraint Handling Approach

In order for PSO to handle constrained problems, it would need to be modified [59]. By the preservation of feasible solutions, particles can search the whole design space but keep track of the feasible solutions only. Two conditions are introduced into the algorithm such that the best feasible solution of a particle is preserved as an individual particle's best solution and the best feasible solution throughout the whole swarm is considered as the global best solution. The above method is applied in this study through the tracking of constraint violations. In this current research, the following conditions are applied for handling constraints:

- Condition 1: Any feasible solution is chosen over an infeasible solution.
- Condition 2: Between two feasible solutions, the one with the better objective function value is preferred.

By applying the first condition, the algorithm orients itself to favor feasible solutions over infeasible solutions and by applying the second condition, the algorithm directs the search towards the feasible regions with good solutions [60]. An illustration of the proposed approach for constraint handling is displayed in Figures 2-4 and 2-5 where the circular grey region highlighted as feasible space contains all possible feasible solutions in the design space and the rest of the design space consists of infeasible solutions. As can be seen in Figure 2-4, $f(\mathbf{x}^{(i,k+1)}) < f(\mathbf{x}^{(i,k)})$ however $\mathbf{x}^{(i,k+1)}$ is not a feasible solution therefore $\mathbf{x}^{(i,k)}$ is retained by the algorithm as the best

Figure 2-4 Schematic Diagram of the Constraint Handling Approach - 1st Condition

solution. In Figure 2-5 since $\mathbf{x}^{(i,k+1)}$ is a feasible solution and $f(\mathbf{x}^{(i,k+1)}) < f(\mathbf{x}^{(i,k)})$, $\mathbf{x}^{(i,k+1)}$ is retained by the algorithm. Reinitialization of particles' initial position until all are within the feasible region was suggested as a method [59] to avoid premature convergence in an infeasible region, however it was observed in this study that this approach would increase the computational time and therefore another method was applied to compensate for this issue. By applying a penalty on unfeasible solutions [61], PSO avoids premature convergence when coupled with the abovementioned conditions. In this study, two pieces of information are included in the penalty function [62] – the number of constraint violations and the amount by which the constraint was violated. Therefore, the modified PSO algorithm solves the cost function as such:

$$F_i(\mathbf{x}) = \begin{cases} f_i(\mathbf{x}) & \text{if feasible solution} \\ f_i(\mathbf{x}) + \beta_1 \left(\sum_{i=1}^d h \right) + \beta_2 \left(\sum_{i=1}^d y \right), & \text{if infeasible solution} \end{cases} \quad 2.10$$

where $F_i(\mathbf{x})$ is the penalty function, $f_i(\mathbf{x})$ is the original cost function, $\sum_{i=1}^d h$ is the sum of d violated constraints, $\sum_{i=1}^d y$ is the sum of the amount of d violated constraints and β_1 and β_2 are the penalty factors. Both β_1 and β_2 are set to the value of 10^5 throughout this research work.

Figure 2-5 Schematic Diagram of the Constraint Handling Approach - 2nd Condition

2.4 Modified Variants of PSO

Many researchers attempted to improve the search capabilities, convergence speed and quality of results by modifying the original PSO algorithm. Several variants of PSO have been developed either by merging concepts of PSO with other search methods or by introducing novel concepts to

the PSO algorithm. Solving complex optimization problems, such as dynamic optimization problems, integer programming problems and multi-objective optimizations problems, essentially requires modifications in the original PSO algorithm. The following lists some modified variants of the original PSO algorithm:

- **Cognition – only** [63]: This version excludes the social component vector from the original particle velocity-update equation. Due to lack of global information, the particles hover around their best positions only. It was noted that this version was more susceptible to failure than the original version and that it required more iterations to converge.
- **Social – only** [63]: In this version, the cognitive component vector from the original particle velocity-update equation is excluded. Here, the particles have no tendency to return to their best historical positions. Particles are only attracted to the best global solution. When empirical comparisons were made between this version with the cognition-only and the original PSO version, it was illustrated that the social-only version had a faster convergence speed and that it was more efficient.
- **Constriction Model** [64]: An alternative to the inertia weight, the constriction coefficient was developed to balance the local and global search trade – off by damping all vector components in the particle velocity-update equation. With this factor, velocity clamping is not needed and convergence is forced. In another study [51], it was shown empirically that by clamping the velocity in the constriction model, quicker convergence was obtained.
- **Fully Informed Model** [36]: In the original PSO version, a particle learns socially from the global leader only while disregarding any information from the other particles. In the fully informed model, all particles learn from each other, i.e. particles update their velocity

by using the best positions of all other particles. Results from numerical experiments demonstrated the superiority of this version compared to the original version.

- **HPSO – TVAC** [41]: In this variant, some previously developed concepts are merged together with some novel modifications. Here, the momentum-less PSO version is coupled with a linearly decreasing inertia weight and time varying acceleration coefficients. To prevent static behaviour of particles, a reinitialization strategy is applied to a particle's velocity through a time-based scheme. The performance of HPSO-TVAC was validated using some benchmark problems where it was demonstrated that it managed to outperform the other methods considered.
- **CPSO** [65]: In this variant, a co-evolutionary particle swarm optimization system for solving constrained problems was introduced. Here, two swarms are generated – one for determining the optimal design variables and the other for finding appropriate penalty factors. Numerical results based on multiple constrained optimization problems demonstrated satisfactory results for the CPSO, when compared to previously reported results.
- **HGAPSO** [66]: A hybrid between GA and PSO, known as HGAPSO, creates a population of individuals by the processes of mutation, crossover and elitism. The concept of elitism is adopted in this variant, where the top-half of the best performing individuals within a population are considered as elites. Before passing on to the next iteration, enhancement of the elites takes place. The group containing the elites are regarded as a swarm of particles, this swarm undergoes enhancement through PSO. In the new iteration, the enhanced elites comprise half of the population while the other half contains individuals generated by applying the crossover and mutation operators on the enhanced elites. The

incorporation of GA and PSO imitates the natural occurring phenomena of maturation and it was shown in the study that it managed to outperform both GA and PSO.

- **IPSO** [67]: A method known as the Improved Particle Swarm Optimization (IPSO) was developed by creating multiple sub-swarms from a population of randomly distributed particles. An individual sub-swarm evolves based on PSO and at a certain predefined time, particles are shuffled and sub-swarms are reassigned. This process ensures that particles are learning new information. According to the study, this method managed to enhance the exploitation and exploration abilities in the swarm. Simulation results for some considered benchmark problems showed that IPSO managed to outperform the original PSO algorithm.
- **PSACO** [68]: In this hybrid variant of PSO and ACO, common characteristics from both methods are merged to produce a two-stage algorithm. In the primary stage, PSO is utilized for global exploration of the design space and in the secondary stage ACO applies a localized search in which ants make use of the mechanism of pheromone trails to refine the particles' position obtained in the first stage. Based on numerical simulations of several benchmark optimization problems, the study concluded that the results demonstrated the efficiency and effectiveness of PSACO in comparison to the considered metaheuristic methods.
- **Dynamic Optimization**: A method [69] was proposed wherein by occasionally resetting both the individual particles' and the swarm's best solution, particles can keep track of the changing global solution. This is achieved by triggering the resetting process either at regular time intervals or by detecting changes in objective function. This approach ensures that the particles forget past and outdated information and direct itself towards the new

global solution. In another study [52], successful tracking and optimization of a dynamic system was achieved by using a random inertia weight in the range of $[0.5, 1.0]$ with a mean of 0.75.

- **Discrete Optimization:** Solving binary problems was the motive for developing the first discrete version of PSO [70]. In this version, the particle velocity-update equation functions as a probability threshold to determine whether a dimensional component in the particle position should be evaluated as a 0 or a 1. This was made possible by using the sigmoid function, defined as $\text{sig}(x) = 1/(1 + \exp(-x))$. Comparison between $\text{sig}(x)$ and a randomly generated number determines the value of x . If the random number is less than $\text{sig}(x)$, x takes the value of 1 otherwise as 0. In a study [71], it was demonstrated that by simply truncating real values generated by PSO into integer values, the performance of PSO was not affected. By default, PSO operates across a continuous design space. Empirical results comparing the performance of the truncation method with the Branch and Bound (BB) technique illustrated the ability of PSO to, efficiently, handle integer problems, and it was shown that PSO managed to outperform the BB technique.
- **Multi-Objective Optimization:** In one of the earliest variants of PSO [72], a concept called dynamic neighborhood was developed for solving multi-objective optimization problems. This approach allows solving two objective functions simultaneously only. At every iteration, a particle defines its neighborhood by calculating its distance from the other particles and based on the distance, a particle selects its neighbors. Here, the difference between the fitness of each particle with respect to one of the objective functions describes the distance. After establishing the neighborhood, the next step involves finding the best local solution of the second objective function among the members of the neighborhood.

Another variant known as MOPSO [73] makes use of Pareto dominant solutions. In this approach, the algorithm creates a second population containing the nondominated solutions found thus far. Next, the algorithm randomly selects a best neighborhood solution from the second population to update the velocities of the particles in the primary swarm. Furthermore, this approach introduced an adaptive grid for the generation of well distributed Pareto fronts and mutation operators to enhance the global search capabilities of the particles.

2.5 Summary

In this chapter, the concept of the particle swarm optimization was introduced and its control parameters were studied to understand how they influence PSO. Due to the lack of a mechanism in handling constraints, modifications need to be enforced in PSO. Here, an approach was proposed based on retaining feasible solutions and utilizing a penalty-based scheme. By forcing the particles to update their best solutions according to its feasibility, they are being oriented to navigate to feasible regions. Furthermore, a discussion was provided to cover some modified PSO variants.

Chapter 3: Parameter Sensitivity Analysis

Dependency of the PSO method with respect to its initial control settings was demonstrated in literature in regards to an unconstrained environment [31, 50, 74]. In this research, a parametric sensitivity analysis is conducted to study the impact of the control parameters in influencing the performance of the algorithm in a constrained environment. Maximum number of iterations, swarm size, inertia weight, the acceleration coefficients and the particle velocity clamping-limit are the control setting parameters in PSO. Poor configuration of the control settings may lead to an increase in the likelihood of premature convergence occurring [43]. A simple sensitivity measure approach known as the one at a time approach is applied here to study the effect produced in the output by changing the value of a single parameter [75, 76]. The sensitivity measure is obtained by increasing a parameter with a given increment while leaving the other parameters fixed and then quantifying the output change in the model. This approach allows the user to know which parameter is responsible for the failure of the model.

3.1 Parameter Sensitivity Analysis – Case Study

Figure 3-1 Schematic Representation of the Speed-Reducer Design Problem

The speed-reducer optimization problem [77] is a common benchmark example used for testing and comparing optimization methods. Here it is considered for the sensitivity analysis. In this problem, the objective is to minimize the weight of the speed-reducer while optimizing seven design variables – face width (b), module of teeth (m), number of teeth in the pinion (z), length of the first shaft between bearings (l_1), length of the second shaft between bearings (l_2), and the diameter of first (d_1) and second shafts (d_2), respectively. All design variables are continuous apart from z which is an integer value. The integer variable is handled using the truncated method [71]. Furthermore, the problem is constrained by 11 constraints. Description of the constrained optimization problem is given below [77]:

$$\begin{aligned} \min f(\mathbf{x}) = & 0.7854x_1x_2^2(3.3333x_3^2 + 14.9334x_3 - 43.0934) - 1.508x_1(x_6^2 + x_7^2) \\ & + 7.4777(x_6^3 + x_7^3) + 0.7854(x_4x_6^2 + x_5x_7^2) \end{aligned} \quad 3.1$$

$$c(\mathbf{x})_1 = \frac{27}{x_1x_2^2x_3} - 1 \leq 0 \quad 3.2$$

$$c(\mathbf{x})_2 = \frac{397.5}{x_1x_2^2x_3^2} - 1 \leq 0 \quad 3.3$$

$$c(\mathbf{x})_3 = \frac{1.93x_4^3}{x_2x_6^4x_3} - 1 \leq 0 \quad 3.4$$

$$c(\mathbf{x})_4 = \frac{1.93x_5^3}{x_2x_6^4x_3} - 1 \leq 0 \quad 3.5$$

$$c(\mathbf{x})_5 = \frac{\sqrt{745\left(\frac{x_4}{x_2x_3}\right)^2 + 16.9 \times 10^6}}{110x_6^3} - 1 \leq 0 \quad 3.6$$

$$c(\mathbf{x})_6 = \frac{\sqrt{745\left(\frac{x_5}{x_2x_3}\right)^2 + 157.5 \times 10^6}}{85x_7^3} - 1 \leq 0 \quad 3.7$$

$$c(\mathbf{x})_7 = \frac{x_2x_3}{40} - 1 \leq 0 \quad 3.8$$

$$c(\mathbf{x})_8 = \frac{5x_2}{x_1} - 1 \leq 0 \quad 3.9$$

$$c(\mathbf{x})_9 = \frac{x_1}{12x_2} - 1 \leq 0 \quad 3.10$$

$$c(\mathbf{x})_{10} = \frac{1.5x_6 + 1.9}{x_4} - 1 \leq 0 \quad 3.11$$

$$c(\mathbf{x})_{11} = \frac{1.1x_7 + 1.9}{x_5} - 1 \leq 0 \quad 3.12$$

where the ranges of the design variables are formulated as follows:

$$\begin{aligned} 2.6 \leq x_1 \leq 3.6, \quad 0.7 \leq x_2 \leq 0.8, \quad 17 \leq x_3 \leq 28, \quad 7.3 \leq x_4 \leq 8.3, \quad 7.3 \leq x_5 \leq 8.3, \\ 2.9 \leq x_6 \leq 3.9, \quad 5.0 \leq x_7 \leq 5.5. \end{aligned}$$

3.2 Numerical Experiment Setup

Each control parameter will be studied individually in order to determine the sensitivity of the modified parameter. The problem is run 100 times, in PSO, to extract statistical information by storing the best global solution at the end of each run. The best global solutions are compiled to identify the best and worst solutions across the 100 runs. The mean value and standard deviation are calculated by the following equations respectively:

$$f(\mathbf{x}_M) = \frac{\sum_{i=1}^{100} f(\mathbf{x}_G^i)}{100} \quad 3.13$$

$$SD = \sqrt{\frac{\sum_{i=1}^{100} (f(\mathbf{x}_G^i) - f(\mathbf{x}_M))^2}{100 - 1}} \quad 3.14$$

where $f(\mathbf{x}_M)$ is the mean value obtained across 100 runs, $f(\mathbf{x}_G^i)$ is the best global solution obtained at run i and SD is the standard deviation. The constraint handling technique described in the previous chapter is applied on the speed-reducer optimization problem. The following numerical experiment was formulated using the numerical computing software MATLAB (R2017a – Academic Version) and the simulations were run on Intel Core i7 with 16 GB RAM. The MATLAB code for the Speed-Reducer Design Problem is provided in the Appendices.

3.3 Numerical Experiment Results

The results of the above problem are displayed in the following subsections for the five control parameters: maximum number of iterations, swarm size, inertia weight, acceleration coefficients and clamping-limit.

3.3.1 Maximum Number of Iterations

The maximum number of iterations depends on the nature of the problem. A low value decreases the probability of obtaining the global solution whereas a high value increases the computational demands. The effect of changing the maximum number of iterations is studied, here, by incrementally increasing the number of iterations by 200 up to 2000 iterations and fixing the other control parameters with the following values:

- Swarm Size: 50
- Inertia Weight: 0.6
- Accelerations coefficients: $c_1 = c_2 = 2$
- Clamping-Limit: $V_{max} = 0.1(x_{max})$

Table 3-1 Mean Value and Standard Deviation (in brackets) with Varying Maximum Number of Iterations

Max. No. of Iterations	Mean Value (Standard Deviation)
200	3.008 E+03 (18.61)
400	3.002 E+03 (12.70)
600	3.002 E+03 (12.11)
800	3.003 E+03 (14.34)
1000	3.001 E+03 (12.34)
1200	3.001 E+03 (12.37)
1400	3.001 E+03 (13.37)
1600	3.002 E+03 (12.73)
1800	3.002 E+03 (13.08)
2000	3.001 E+03 (11.61)

Increasing the maximum number of iterations increases the possibility of PSO locating the global optimum or a local solution close to it. In Table 3-1, it can be seen that a lower mean value was reported by PSO when the iterations was increased from 200 to 400 and in subsequent iterations a similar performance was produced. PSO managed to locate solutions close to the vicinity of the global minima at iterations higher than 200 and this can be demonstrated through Figure 3-2 where a lower worst value was reported at iterations 400 to 2000.

Figure 3-2 Performance (Best and Worst Value) of PSO with Varying Maximum Number of Iterations

3.3.2 Swarm Size

It was claimed in a study [78], where the sensitivity of the swarm size was investigated, that swarm sizes have minimal effect in relation to the performance of PSO. Here, the swarm size is studied by incrementally increasing the number of particles by 10 up to 100 particles and fixing the other control parameters with the following values:

- Maximum No. of Iterations: 600
- Inertia Weight: 0.6
- Accelerations coefficients: $c_1 = c_2 = 2$
- Clamping-Limit: $V_{max} = 0.1(x_{max})$

The claim made in the empirical study [77] is accurate in terms of performance, i.e. obtaining the best value, as PSO was able to obtain the global optimum even with 10 particles. However, as can be in Figure 3-3, the quality of results improved with more particles in the design space as the worst value decreased. Moreover, in terms of efficiency and robustness, the swarm size played a very important role as the mean value moved closer to the global minima as the swarm size increased and coupled with a decreasing standard deviation led to a higher confidence in the performance of PSO as seen in Table 3-2.

Figure 3-3 Performance (Best and Worst Value) of PSO with Varying Swarm Sizes

Table 3-2 Mean Value and Standard Deviation (in brackets) with Varying Swarm Sizes

Swarm Size	Mean Value (Standard Deviation)
10	3.039 E+03 (43.49)
20	3.021 E+03 (21.88)
30	3.010 E+03 (18.20)
40	3.004 E+03 (14.02)
50	3.002 E+03 (12.11)
60	3.001 E+03 (11.95)
70	3.000 E+03 (9.33)
80	2.999 E+03 (8.24)
90	2.996 E+03 (5.69)
100	2.996 E+03 (4.98)

3.3.3 Inertia Weight

Figure 3-4 Dynamic Behavior of the Considered Inertia Weight Variants

Inertia weight dictates the influence of the previous momentum to the current velocity. Both constant and dynamic inertia weight are considered here. For the constant inertia, multiple values are selected by increments of 0.1 starting from 0 up to 1. As for the dynamic inertia weight, the linearly decreasing (LDIW) [50], nonlinear (NLIW) [54], random (RIW) [52] and chaotic (CIW) [55] inertia weight are included in this study. The dynamic behavior of the considered inertia weight variants is displayed in Figure 3-4. Therefore, the momentum-less ($w = 0$) [50] and the original PSO ($w = 1$) versions are also studied. NLIW, LDIW, CIW and the rest of the constant control parameters are set as:

- NLIW [54]: $n = 0.9$, $w_{min} = 0.4$ and $w_{max} = 0.9$
- LDIW [50]: $w_{min} = 0.4$ and $w_{max} = 0.9$
- Maximum No. of Iterations: 600
- Swarm Size: 50
- Accelerations coefficients: $c_1 = c_2 = 2$
- Clamping-Limit: $V_{max} = 0.1(x_{max})$

Table 3-3 Best, Worst, Mean and Standard Deviation (in brackets) with Varying Inertia Weight

Inertia Weight	Best Value	Worst Value	Mean Value (Standard Deviation)
0.0	2.994 E+03	2.994 E+03	2.994 E+03 (1.36 E-07)
0.1	2.994 E+03	3.007 E+03	2.995 E+03 (3.20)
0.2	2.994 E+03	3.007 E+03	2.996 E+03 (3.73)
0.3	2.994 E+03	3.007 E+03	2.997 E+03 (4.65)
0.4	2.994 E+03	3.034 E+03	2.998 E+03 (7.03)
0.5	2.994 E+03	3.043 E+03	3.000 E+03 (11.01)
0.6	2.994 E+03	3.047 E+03	3.002 E+03 (12.11)
0.7	2.994 E+03	3.047 E+03	3.006 E+03 (16.03)
0.8	2.995 E+03	3.047 E+03	3.014 E+03 (19.18)
0.9	2.996 E+03	3.057 E+03	3.028 E+03 (18.90)
1	3.010 E+03	3.059 E+03	3.038 E+03 (16.03)
LDIW	2.994 E+03	3.056 E+03	3.011 E+03 (19.87)
NLIW	2.994 E+03	3.056 E+03	3.011 E+03 (19.73)
RIW	2.994 E+03	3.047 E+03	3.010 E+03 (17.31)
CIW	2.994 E+03	3.047 E+03	3.005 E+03 (14.57)

Table 3-3 shows the results of the sensitivity analysis on the inertia weight. Interestingly the data obtained here show that at certain values, between 0 ~ 0.6, the constant inertia weight managed to outperform the dynamic variants of the inertia weight, which is in contrary to multiple studies [50, 79]. As for the performance of the dynamic inertia weights, CIW managed to perform best due to the increased randomness introduced to the particles, thereby promoting diversity in the search leading to a more robust search.

Figure 3-5 Comparison of the Momentum-Less PSO and the Chaotic Inertia Weight PSO in Terms of Convergence

In Figure 3-5, the convergence curves of the optimal solution display that the momentum-less version managed to converge to the global minimum with fewer iterations and thus performed more efficiently than the CIW version.

3.3.4 Acceleration coefficients

In a study [80] attempting to understand the behaviour of the particles in PSO, the author suggested that the sum value of both the cognitive and social component in the particle velocity-update equation should not exceed 4. Therefore, it was established from the original conception of PSO to assign a value of 2 for each of the acceleration coefficients [1]. In this section, multiple combinations of the acceleration coefficient are studied – all combinations have a sum of 4. The social – only ($c_1 = 0$ and $c_2 = 4$) [63] and original PSO ($c_1 = 2$ and $c_2 = 2$) [1] versions are considered here for comparison as well. Furthermore, the time varying acceleration coefficients (TVAC) is also included in this study to examine whether the promotion of a global exploration in the early iterations and a local exploitation in the latter stages can successful navigate the particles in finding the best solution [41]. The fixed parameters are thus set:

- TVAC [41]: $c_{max} = 2.5$ and $c_{min} = 0.5$
- Maximum No. of Iterations: 600
- Swarm Size: 50
- Inertia Weight: 0.6
- Clamping-Limit: $V_{max} = 0.1(x_{max})$

Table 3-4 Best, Worst, Mean and Standard Deviation (in brackets) with Varying Acceleration Coefficients

Acceleration Coefficients	Best Value	Worst Value	Mean Value (Standard Deviation)
$c_1 = 0, c_2 = 4$	3.007 E+03	3.364 E+03	3.151 E+03 (81.28)
$c_1 = 0.5, c_2 = 3.5$	2.994 E+03	3.198 E+03	3.041 E+03 (45.59)
$c_1 = 1, c_2 = 3$	2.994 E+03	3.056 E+03	3.012 E+03 (19.64)
$c_1 = 1.5, c_2 = 2.5$	2.994 E+03	3.047 E+03	3.005 E+03 (14.21)
$c_1 = 2, c_2 = 2$	2.994 E+03	3.047 E+03	3.002 E+03 (12.11)
$c_1 = 2.5, c_2 = 1.5$	2.994 E+03	3.034 E+03	3.000 E+03 (11.17)
$c_1 = 3, c_2 = 1$	2.994 E+03	3.034 E+03	2.998 E+03 (9.10)
$c_1 = 3.5, c_2 = 0.5$	2.994 E+03	3.050 E+03	3.007 E+03 (16.72)
TVAC	2.994 E+03	2.994 E+03	2.994 E+03 (6.83 E-06)

Sensitivity analysis of the acceleration coefficients is tabulated in Table 3-4. The results show that the conventional setting of the acceleration coefficient does not necessarily provide the best performance. The cognitive – only [63] ($c_1 = 4$ and $c_2 = 0$) version was originally included however due to the static behavior of the particles, they were not able to explore the design space due to the absence of a global leader for attraction and thus performed very poorly. In addition, the results show that the original PSO version managed to perform better than the social-only version; this is in contrary to an empirical study [63]. Finally, the dynamic behavior of TVAC managed to navigate the particles from an explorative search to an exploitative search resulting in a strong performance.

Figure 3-6 Comparison of the Convergence Curves of the Conventional Acceleration Coefficient and TVAC

In Figure 3-6, the convergence curve of the conventional acceleration coefficients is compared with the TVAC. It can be seen that conventional method led to a quick convergence whereas TVAC required more iterations as it directed particles to explore the design space in the initial iterations and then towards the end of the search TVAC promoted convergence to the global solution.

3.3.5 Clamping-Limit

In order to control global exploration and to ensure that particles do not stray far from the global and best solution, the velocities are clamped to be bounded [81]. In this sensitivity analysis study, multiple values of clamping-limits are studied from 0 to 1 with increments of 0.1. A linearly decreasing clamping-limit (LDCL) was suggested by a couple of studies [44, 82] and is applied here to investigate whether a dynamic clamping-limit could be successful in enhancing the search capabilities of PSO. It is described with the following form:

$$V_{max} = \left(\frac{(V_{max}^U - V_{max}^L) \times (Max_k - k)}{Max_k} + V_{max}^L \right) \quad 3.15$$

where V_{max}^U is the upper limit of the velocity limit and V_{max}^L is the lower limit of the velocity limit.

The fixed parameters are thus set:

- LDCL: $V_{max}^U = 1$ and $V_{max}^L = 0.1$
- Maximum No. of Iterations: 600
- Swarm Size: 50
- Inertia Weight: 0.6
- Accelerations coefficients: $c_1 = c_2 = 2$

Table 3-5 Best, Worst, Mean and Standard Deviation (in brackets) with Varying Clamping-Limit

Clamping-Limit	Best Value	Worst Value	Mean Value (Standard Deviation)
0.1	2.994 E+03	3.047 E+03	3.002 E+03 (12.11)
0.2	2.994 E+03	3.047 E+03	3.003 E+03 (14.03)
0.3	2.994 E+03	3.047 E+03	3.003 E+03 (14.31)
0.4	2.994 E+03	3.047 E+03	3.005 E+03 (16.90)
0.5	2.994 E+03	3.047 E+03	3.003 E+03 (14.36)
0.6	2.994 E+03	3.047 E+03	3.002 E+03 (13.69)
0.7	2.994 E+03	3.047 E+03	3.002 E+03 (14.21)
0.8	2.994 E+03	3.047 E+03	3.001 E+03 (13.84)
0.9	2.994 E+03	3.047 E+03	3.000 E+03 (12.25)
1.0	2.994 E+03	3.047 E+03	3.001 E+03 (13.49)
LDCL	2.994 E+03	3.047 E+03	3.002 E+03 (13.78)

The results produced show that the clamping-limits have minimal impact on the algorithm's performance; LDCL performs in a likewise manner. However, this behavior might be due to the nature of the problem as the maximum velocity is commonly set to the dynamic range of the design variables [25, 52]. The empirical results of the sensitivity analysis of the clamping-limit on the PSO algorithm are displayed in Table 3-5.

3.4 Summary

Based on the results produced across all sensitivity measure cases, it can be seen that the PSO algorithm is sensitive to the control parameters and that its performance depends on the initial settings. With respect to the maximum number of iterations, the empirical results revealed that each problem is unique – different problems require a different number of iterations. Therefore, initiating the algorithm with a large number of iterations can be redundant. More diversity in the population led to a better quality of results, however by increasing the swarm size, more computational effort is induced. Also, the results showed that the impact of the clamping-limit on PSO's performance was minimal.

Overall, the sensitivity analysis study demonstrated that the PSO algorithm is most sensitive to the inertia weight and the acceleration coefficients. By optimally tuning the inertia weight and acceleration coefficients, PSO would be able to balance the global exploration and the local exploitation and overcome the need to initiate the algorithm with a large swarm size.

Chapter 4: Proposed Method and Its Implementation

In the previous chapter, the sensitivity analysis of the control parameters in PSO revealed that PSO is highly dependent and sensitive to its initial settings; in particular, the inertia weight and acceleration coefficients have the most impact on its performance. A number of variants of the inertia weight have been developed and some were considered in the sensitivity analysis, where it was demonstrated that they were ineffective in navigating the particles – resulting in a poor performance. Many researchers advocated the need for the inertia weight to be large in the beginning and then gradually decrease to a minimum value [50, 78]. However, the variation of the inertia weight based on time may not be effective and thus a self – adapting inertia weight would be optimal [78]. In this chapter, a scheme of control parameter adaption is proposed.

4.1 Adaptive Particle Swarm Optimization (APSO)

Sharing common behavior such as being population based and stochastic, the PSO method, like the other nature – inspired optimization methods, could be ineffective, as at times it requires a very high number of function evaluations to find the optimal solution [83]. This ineffectiveness is further evident in multimodal and high dimensional problems [84]. Thus, a need to introduce modifications to the PSO algorithm is present in order to promote convergence speed and avoid premature convergence. As previously mentioned in Chapter 2, a number of variants of the PSO algorithm have been developed [41, 65, 66, 67] to solve the aforementioned issues present in PSO. In this chapter, an adaptive particle swarm optimization (APSO) is devised by developing a feedback system that adapts the inertia weight and the two acceleration coefficients through observing the environment and modifying the three control parameters to help navigate the particles in finding the global optimum in fewer iterations. The measure by which the considered strategy is implemented should be easily computable and be effective in directing the particles

successfully [85]. This paper proposes the evolutionary state as a novel means to adapt the PSO algorithm so that it can overcome its limitations by measuring the performance of a particle's current position with respect to its personal best solution and the swarm's best solution.

4.1.1 Motivation

A unique $c_1^{(i,k)}$, $c_2^{(i,k)}$ and $w^{(i,k)}$ for each particle is the goal of the proposed scheme. This unique property is self-automated based on the feedback parameter generated by the fitness of each particle. Many variations of PSO based on adapting strategies exist using different sources as the feedback parameter to track and monitor the condition of the algorithm and make changes according to the circumstances. In one version [86], $w^{(i,k)}$ is adapted based on two feedback parameters called the speed factor and the aggregation degree factor. The speed factor measures whether a particle managed to improve its personal best solution, whereas the aggregation degree factor compares the mean performance of all particles with the best performer in the current iteration. Applying the speed and aggregation factors, $w^{(i,k)}$ is determined as follows:

$$w^{(i,k)} = w_{initial} - \alpha(1 - h^{(i,k)}) + \beta S \quad 4.1$$

where $w_{initial}$ is the initial value of the inertia weight, $h^{(i,k)}$ is the speed factor, S is the aggregation factor, α and β are two system components typically set within the range of [0,1]. Some other researchers opted to use the fitness value of particles as the characteristic component in adapting the inertia weight of each particle. One such modification adapts the inertia weight by calculating the ratio of the global best solution to the average fitness value of all personal best solutions [87]. In another study [85], an adapting strategy based on the success of particles was proposed. A particle is considered successful at a certain iteration if it managed to achieve a new personal best solution, otherwise it is considered unsuccessful. The inertia weight using the success percentage is computed as follows:

$$w^{(i,k)} = w_{min} + (w_{max} - w_{min}) \left(\frac{\sum_{i=1}^N S^{(i,k)}}{N} \right) \quad 4.2$$

where $S^{(i,k)}$ is 1 if successfully and 0 if unsuccessful and N represents the swarm size. As the success percentage increases, the inertia weight increases as well and as the success percentage decreases so does the inertia weight. By splitting the search into four states – exploration, exploitation, convergence and jumping out, some researchers were able to adaptively adjust PSO using the Euclidean distance of each particle as the feedback parameter [88]. Based on a fuzzy system, the acceleration coefficients are adjusted and the inertia weight updated according to the feedback parameter. Furthermore, a mutation operator was applied to the global leader to help promote a refined search and avoid premature convergence. However, all of the above-mentioned adaptive variants of PSO have been designed to solve unconstrained problems. Therefore, in the proposed method, the constraint handling technique works in conjunction with the adapting scheme to ensure that the particles are adapting to the environment by directing itself to the feasible regions.

4.2 Unique Adaptive Particle Swarm Optimization (UAPSO)

The proposed APSO method is hereby termed as Unique Adaptive Particle Swarm Optimization (UAPSO). In UAPSO, each particle has a unique cognitive parameter $c_1^{(i,k)}$, social parameter $c_2^{(i,k)}$ and inertia weight $w^{(i,k)}$. As mentioned earlier, the evolutionary state is the feedback parameter used as the mechanism to self-adapt the algorithm based on the environment. The evolutionary state (ES) is given by:

$$ES^{(i,k)} = \frac{f(\mathbf{x}_p^{(i,k)}) - f(\mathbf{x}_g^{(k)})}{f(\mathbf{x}_f^{(i,k)})} \quad 4.3$$

where $f(\mathbf{x}_p^{(i,k)})$ is the fitness of each individual's personal best solution, $f(\mathbf{x}_G^{(k)})$ is the fitness of the best historical solution across all particles and $f(\mathbf{x}_f^{(i,k)})$ is the fitness of each particles latest or most recent feasible solution. Since any given constrained problem involves both feasible and infeasible solutions, in UAPSO the particles adapt solely according to the feasible solutions in the design space and this promotes better learning within the swarm. This is implemented by applying a condition to Equation 4.3, whereby only feasible solutions are accepted. Thus, the following equation demonstrates the selection process by which $f(\mathbf{x}_f^{(i,k)})$ is determined:

$$f(\mathbf{x}_f^{(i,k)}) = \begin{cases} f(\mathbf{x}_p^{(i,k)}), & \text{if } k = 1 \\ f(\mathbf{x}^{(i,k)}), & \text{if } \mathbf{x}^{(i,k)} \text{ is a feasible solution} \\ f(\mathbf{x}_f^{(i,k)}), & \text{if } \mathbf{x}^{(i,k)} \text{ is a infeasible solution} \end{cases} \quad 4.4$$

In the first iteration, $f(\mathbf{x}_f^{(i,k)})$ is set to that of the best personal solution for all particles. This is done since in the beginning feasible solutions cannot be guaranteed according to the established constraint handling technique. In the following iterations ($k > 1$), whenever a feasible solution is obtained it is designated as the latest feasible solution and in cases where an infeasible solution is obtained, the latest feasible solution would remain as the current $f(\mathbf{x}_f^{(i,k)})$. Furthermore, since a particle's best solution and the swarm's best solution are updated according to their feasibility, the evolutionary state provides a feedback that is reliable as it includes factors that are both feasible and easily computable.

4.2.1 UAPSO – Inertia Weight

The evolutionary state acts as the inertia weight, $w^{(i,k)}$, in the proposed method. As can be understood from Equation 4.3, whenever the value of $ES^{(i,k)}$ is high the latest feasible solution of a particle is close to its personal best and far away from the global best solution. Whenever the value of $ES^{(i,k)}$ is low, it either means (1) that the latest feasible solution of the particle is far away

from both its personal best and the global leader or (2) the particle's personal best solution is close to the global leader. In order to depict the two cases, a simple illustration representing a single dimension minimization problem is displayed in Figures 4-1 and 4-2. In cases where $f(x_f^{(i,k)}) = f(x_p^{(i,k)})$, a high $ES^{(i,k)}$ means that the particle is far away from the global leader and in the condition when $ES^{(i,k)}$ is low, the particle is close to the global leader. Furthermore, whenever $f(x_p^{(i,k)}) = f(x_g^{(k)})$, the evolutionary state of the particle is equal to 0.

Figure 4-1 Schematic description of low Evolutionary State when $f(x_f^{(i,k)}) \neq f(x_p^{(i,k)})$ or/and $f(x_f^{(i,k)}) \neq f(x_g^{(k)})$

Figure 4-2 Schematic description of high Evolutionary State when $f(x_f^{(i,k)}) \neq f(x_p^{(i,k)})$ or/and $f(x_f^{(i,k)}) \neq f(x_g^{(k)})$

Influence of a particle's momentum is dependent on the value of its inertia weight, the higher the value of $w^{(i,k)}$ the greater the influence of a particle's momentum and the lower the value of $w^{(i,k)}$ the lesser the influence of a particle's momentum. Thus, at lower $w^{(i,k)}$ particles are more attracted towards their best personal and the swarm's best solution and attempts to converge towards those solutions and at higher $w^{(i,k)}$, particles makes use of their momentum to explore more regions and attempts to discover new optimal solutions. The value of $w^{(i,k)}$ fluctuates between $[0, 1]$. As an example, the dynamics of the inertia weight in UAPSO is illustrated in Figure 4-3.

Figure 4-3 Dynamic Behavior of the Inertia Weight in UAPSO

4.2.2 UAPSO – Acceleration Coefficients

In the original PSO algorithm [1], a constant value of 2 for both acceleration coefficients was established and in a subsequent research [40], it was found that a linear dynamic form proved more successful and this was also demonstrated in the sensitivity analysis performed in this research work. Due to the superior performance of the dynamic form, it is applied in the proposed method. However, instead of being purely based on time, in UPASO it is also influenced by the evolutionary state. A particle maybe more oriented towards either a local or global search and by allowing the evolutionary state to dictate the acceleration coefficient, particles are promoted to discovering better solutions with respect to their location in the hyperspace. In both the conventional way and TVAC, particles are forced into behaving in a certain way and this could affect their search capabilities. The following equations describes the form of both acceleration coefficients applied in the proposed method:

$$c_1^{(i,k)} = \begin{cases} \left(\frac{(c_{max} - c_{min}) \times (Max_k - k)}{Max_k} \right) + c_{min}, & \text{if } 0 \leq ES^{(i,k)} \leq 0.5 \\ c_{max} - \left(\frac{(c_{max} - c_{min}) \times (Max_k - k)}{Max_k} \right), & \text{if } 0.5 \leq ES^{(i,k)} \leq 1.0 \end{cases} \quad 4.5$$

$$c_2^{(i,k)} = \begin{cases} c_{max} - \left(\frac{(c_{max} - c_{min}) \times (Max_k - k)}{Max_k} \right), & \text{if } 0 \leq ES^{(i,k)} \leq 0.5 \\ \left(\frac{(c_{max} - c_{min}) \times (Max_k - k)}{Max_k} \right) + c_{min}, & \text{if } 0.5 \leq ES^{(i,k)} \leq 1.0 \end{cases} \quad 4.6$$

where $c_{max} = 4$ and $c_{min} = 0$. In this adaption, whenever a particle has a low evolutionary state ($0 \leq ES^{(i,k)} \leq 0.5$), the algorithm promotes a more global search (in the initial iterations) by allowing the particle to explore the regions around its best historical solution and as time goes by a more local search is promoted. In cases where the evolutionary state has a large value ($0.5 \leq ES^{(i,k)} \leq 1.0$), a more exploitive search is promoted (in the initial iterations) by allowing the

particles to converge towards the swarm's best solution and gradually, a more global search is promoted towards the end. Depending on time, at larger evolutionary states particles located far from the global best solution have a higher momentum and based on its acceleration coefficients, are either attracted to its best historical or the global solution. This ensures particles are not being ineffective by allowing them to play a role in obtaining a better solution. At lower evolutionary states, particles explore the design space due to a reduced influence from their respective momentums and the higher influence of either the personal or social component. Illustration of the dynamic behaviour of the acceleration coefficients of a particle are presented in Figure 4-4 where the particle fluctuates between a low and high evolutionary state in the initial 100 iterations after which, due to its low evolutionary state, it gradually shifts from an explorative search to an exploitive search.

Figure 4-4 Dynamic behavior of Acceleration Coefficients – UAPSO

4.2.3 UAPSO – Reinitialization of Velocity

A particle's momentum is essential to its performance since it provides the necessary acceleration to allow particles to roam around the search space and avoid premature convergence. Hence, a reinitialization scheme is implemented with random velocities whenever stagnation occurs. Stagnation may be defined as a phenomenon where no improved solutions are found [39] due to particles being stuck in their respective best found solutions. In the proposed method, reinitialization of a component of the velocity vector with a random velocity occurs whenever stagnation is present in the design space. The reinitialization scheme implemented in the proposed method takes its inspiration from the method applied in the HPSO – TVAC algorithm [41]. Only a single velocity component is reinitialized since the other components in the velocity vector might have a good structure that would enable directing the particle to the global optima or a better local solution. Furthermore, reinitialization of the velocity is with respect to the velocity clamping-limit. The following equation describes the reinitialization scheme applied in UAPSO:

$$v_d^{(i, k+1)} = \begin{cases} rand \times V_{max}^d, & \text{if } v_d^{(i, k+1)} = 0 \text{ and } rand \leq 0.5 \\ rand \times (-V_{max}^d), & \text{if } v_d^{(i, k+1)} = 0 \text{ and } rand > 0.5 \end{cases} \quad 4.6$$

where $v_d^{(i, k+1)}$ refers to a certain dimension d of the velocity vector, $rand$ refers to a random number generated from a uniform distribution in the range $[0, 1]$ and V_{max}^d refers to the clamping limit of dimension d .

4.2.4 UAPSO – Repulsion Based Particle Velocity Update

The original PSO algorithm is susceptible to being trapped into a local solution, which could result in a poor convergence speed and might lead to premature convergence especially for problems with complexity [89]. Some tried to avoid this by introducing a chaotic factor to the velocity-update equation [90] and others by modifying the equation. One such variant is known

as the Repulsive Particle Swarm Optimization (RPSO) and is based on the notion that by introducing repulsion between the particles, global exploration capabilities could be enhanced which would increase the probability of determining the global optima [91]. The velocity update equation in RPSO is modified as follows:

$$\begin{aligned} \mathbf{v}^{(i, k+1)} = & w\mathbf{v}^{(i, k)} + c_1\mathbf{rand}_1^{(k)} \left(\mathbf{x}_p^{(i,k)} - \mathbf{x}^{(i,k)} \right) + c_2\mathbf{rand}_2^{(k)} \left(\mathbf{x}_p^{(j,k)} - \mathbf{x}^{(i,k)} \right) \\ & + c_3\mathbf{rand}_3^{(k)} v_d \end{aligned} \quad 4.7$$

where $\mathbf{x}_p^{(j,k)}$ is the best solution of a randomly selected particle, c_3 is an acceleration coefficient ($c_3 = 0.5$), $\mathbf{rand}_3$ is vector containing random numbers between a range of $[0, 1]$ and v_d is a random component of a particle's velocity. Here $c_2 = -1.43$. The third component in the right-hand side of Equation 4.7 creates a repulsion effect between a random particle's best solution and that of a particle. The third term attempts to navigate the particles in such a way as to avoid being trapped in a local optimum and the fourth term generates noise to enhance exploration across the search area. However, as can be seen in Equation 4.7, the equation lacks any global information and this restricts the overall learning between particles.

In this research work, a fourth term is introduced into the particle velocity-update equation to both enhance the global exploration and increase robustness. By using the evolutionary state, it is possible to adapt the equation based on the proximity of the latest feasible solution to that of the best solutions. The modified particle velocity-update equation is displayed below:

$$\begin{aligned} \mathbf{v}^{(i, k+1)} = & ES^{(i,k)}\mathbf{v}^{(i, k)} + c_1\mathbf{rand}_1^{(k)} \left(\mathbf{x}_p^{(i,k)} - \mathbf{x}^{(i,k)} \right) + c_2\mathbf{rand}_2^{(k)} \left(\mathbf{x}_G^{(k)} - \mathbf{x}^{(i,k)} \right) - (1 \\ & - ES^{(i,k)}) \left(\mathbf{x}_G^{(k)} - \mathbf{x}_p^{(i,k)} \right) \end{aligned} \quad 4.8$$

In Equation 4.8, particles are repulsed based on two factors: (1) the difference between the personal and global best solutions, and (2) the value of the evolutionary state. At higher

evolutionary states, a lower repulsion is experienced and at lower evolutionary states, a higher repulsion is experienced. The pseudocode of the proposed method is displayed in Figure 4-5 and the MATLAB code is provided in the Appendices.

Step 1	Initialization Randomise $x^{(i,0)}$ Set $v^{(i,0)} = 0$ and $x^{(i,0)} = x_p^{(i,0)}$ Determine x_G^0
Step 2	Iterative Cycle $k = 1$: iterations FOR each particle i Calculate the Evolutionary State (Eq. 4.3) Update the Acceleration Coefficients (Eq. 4.4 and Eq. 4.5) Update the Particle Velocity (Eq. 4.8) Update the Particle Location (Eq. 2.2) Calculate objective function $f(x^{(i,k)})$ If $f(x^{(i,k)}) < f(x_p^{(i,k)})$, $x_p^{(i,k)} = x^{(i,k)}$ end IF If $f(x^{(i,k)}) < f(x_G^k)$, $x_G^k = x^{(i,k)}$ end IF If $v_d^{(i, k+1)} = 0$, Reinitialize the Particle's Velocity (Eq. 4.6) end IF end FOR
Step 3	$k = k + 1$ Repeat Step 2 until termination. Termination criteria $k = \text{Max } k$

Figure 4-5 Pseudocode – UAPSO

4.3 Simulations on Benchmark Constrained Optimization Problems

In this section, the performance of UAPSO is examined by solving multiple constrained optimization problems. Four common benchmark-engineering problems that are widely used in literature are considered here to compare the performance of UAPSO with other popular algorithms. These engineering problems include objective functions and constraints with various forms (continuous and mixed integer problems) and structure (linear and nonlinear). Derived optimization results are compared with respect to statistical data and function evaluations, FEs. Here, the statistical data considered include the best, worst and mean function value and the standard deviation. However due to the subtle differences between the results, the statistical data are converted into percentage difference relative to the global solution, while the standard deviation is divided with the mean function value to produce the coefficient of variation (CV). Furthermore, the design variable(s) of the best solution, the constraint(s) value of the corresponding best solution and the number of function evaluations are also included. Function evaluation is calculated as the product of the swarm size and the maximum number of iterations. Convergence speed is evaluated based on the fewest required FEs to obtain the best global solution. The evaluation mechanism, by which the algorithms are compared, is as follows:

1. Algorithms that managed to obtain the best function value are preferred. Any algorithm that did not discover the best global solution is not further considered.
2. The algorithm that determined the best global solution with the fewest function evaluations is preferred over the others.
3. Secondary to the convergence speed, is the robustness and efficiency of the algorithm – this is established by a low coefficient of variation and a low worst and mean performance.

The following numerical simulations were formulated using the numerical computing software MATLAB (R2017a – Academic Version) and the numerical experiments were run on Intel Core i7 with 16 GB RAM. The MATLAB codes of the benchmark problems are provided in the Appendices. The constrained optimization benchmark problems were independently run 100 times to extract the statistical performance of the proposed method. The user parameters applied on the considered benchmark problems are presented in Table 4-1. For each problem, two cases of UAPSO are studied – UAPSO₁ for convergence speed and UAPSO₂ for robustness and efficiency. The maximum velocity is set to the dynamic range of the design variables for all four problems.

Table 4-1 User Parameters for the Benchmark Design Problems

Design Problem		Swarm Size	Max. No of Iterations	FEs
Speed Reducer	UAPSO ₁	5	3,000	15,000
	UAPSO ₂	10	3,000	30,000
Pressure Vessel	UAPSO ₁	5	3,000	15,000
	UAPSO ₂	10	3,000	30,000
Tension/Compression Spring	UAPSO ₁	9	1,000	9,000
	UAPSO ₂	24	1,000	24,000
Welded Beam	UAPSO ₁	10	2,000	20,000
	UAPSO ₂	25	2,000	50,000

Lastly, the considered algorithms in the comparative study range from novel algorithms to hybrid variants of global optimizers. The following lists the algorithms used to study the performance of UAPSO:

- **Hybrid Evolutionary Algorithm and Adaptive Constraint-Handling Technique (HEA-ACT)** [92]: In HEA-ACT, an offspring population is generated by applying a simplex crossover and two mutation operators. Additionally, an adaptive constraint handling technique based on the feasibility of the population is applied.

- **Differential Evolution with Level Comparison (DELC)** [93]: In DELC, a constrained optimization problem is solved by applying a level comparison and using the differential evolution method to determine the global solution.
- **Modified Differential Evolution (MDE)** [94]: In MDE, each parent is allowed to produce more than one offspring and the best performing offspring is selected based on a set of feasibility rules. Apart from this mechanism, MDE utilizes the original mechanism of the differential evolution.
- **Hybrid Particle Swarm Optimization with Differential Evolution (PSO-DE)** [95]: In order to solve premature convergence in PSO, the differential evolution is applied to the best personal solution of each particle obtained within the swarm generated by PSO. Furthermore, constraints are handled by applying feasibility rules.
- **Backtracking Search Algorithm combined with the Self-Adaptive ε constrained method (BSA-SA $_{\varepsilon}$)** [96]: The backtracking search algorithm comprises of two selection processes where former solutions are used in the mutation operation before the application of the crossover process. Additionally, a self-adapting scheme is employed to control the ε value in the ε –constrained method.
- **Backtracking Search Optimization Algorithm Inspired by Simulated Annealing (BSAISA)** [97]: Here, the simulated annealing method is used to adapt the mutation factor in the backtracking search algorithm. Furthermore, a self-adaptive ε –constrained method is employed to handle constraints.
- **Co-Evolutionary Particle Swarm Optimization (CPSO)** [65]: Here, two swarms are generated, one for determining the optimal design variables and the other for finding appropriate penalty factors.

- **Co-Evolutionary Differential Evolution (CDE)** [98]: Similar to CPSO, here two populations are generated, one for adapting penalty factors and the other evolving decision solutions.
- **Genetic Algorithms based on Dominance Tournament-Selection (GA-DT)** [99]: By applying a dominance-based strategy, constraints are incorporated into the objective function of the genetic algorithm.
- **Mine Blast Algorithm (MBA)** [100]: MBA simulates the explosion of mine bombs and aims to locate the mine that would create the biggest explosion. Moreover, a set of feasibility rules are utilized for handling constraints.
- **Hybrid Particle Swarm Optimization (HPSO)** [101]: In this hybrid algorithm, constraints are handled in PSO by applying feasibility rules. In addition, simulated annealing is applied to the swarm's best obtained solution to help avoid premature convergence.

4.3.1 Speed-Reducer

The speed-reducer problem used in the sensitivity analysis is reintroduced in this algorithm comparative study. The best solution obtained by UAPSO is $x = [3.500000, 0.700000, 17.000000, 7.300000, 7.715320, 3.350215, 5.286654]$ with $f(x) = 2994.471066$ in both cases of UAPSO. Comparison between the best solutions obtained by UAPSO with the other algorithms is presented in Table 4-2. The considered algorithms are the HEA-ACT, DELC, MDE, PSO-DE and BSAISA. The statistical performance of the considered methods with respect to the proposed method is presented in Table 4-3. As shown in Table 4-2, all considered methods satisfied the constraints applied in this problem. BSAISA constraint values are not available however, the authors of BSAISA claim that BSAISA satisfied all constraints [97]. In terms of obtaining the best solution, DELC, BSAISA and UPASO managed to obtain the best solution, $f(x) = 2994.471066$, while HEA-ACT, MDE and PSO-DE could not. DELC and BSAISA

required 30,000 FEs and 15,860 FEs respectively, whereas UAPSO₁ required 15,000 only. As for the quality of results, UAPSO₁ performed the worst compared to DELC and BSAISA, however UAPSO₂ managed to perform the best compared to all considered methods. Overall, in terms of convergence speed UAPSO₁ performed the best amongst the considered methods as it managed to obtain the best solution in the fewest FEs. On the other hand, in terms of robustness and effectiveness, UAPSO₂ performed the best across all statistical data outperforming DELC by having a lower coefficient of variation.

Table 4-2 Comparison of the Best Solutions for the Speed-Reducer Problem

D. V.	HEA-ACT	DELC	MDE	PSO-DE	BSAISA	UAPSO
x_1	3.50 E+00	3.50 E+00	3.50 E+00	3.50 E+00	3.50 E+00	3.50 E+00
x_2	7.00 E-01	7.00 E-01	7.00 E-01	7.00 E-01	7.00 E-01	7.00 E-01
x_3	1.70 E+01	1.70 E+01	1.70 E+01	1.70 E+01	1.70 E+01	1.70 E+01
x_4	7.30 E+00	7.30 E+00	7.30 E+00	7.30 E+00	7.30 E+00	7.30 E+00
x_5	7.72 E+00	7.72 E+00	7.80 E+00	7.80 E+00	7.72 E+00	7.72 E+00
x_6	3.35 E+00	3.35 E+00	3.35 E+00	3.35 E+00	3.35 E+00	3.35 E+00
x_7	5.29 E+00	5.29 E+00	5.29 E+00	5.29 E+00	5.29 E+00	5.29 E+00
c_1	-1.98 E-01	-1.98 E-01	-1.98 E-01	-1.98 E-01	N/A	-1.98 E-01
c_2	-4.99 E-01	-4.99 E-01	-4.99 E-01	-4.99 E-01	N/A	-4.99 E-01
c_3	-9.05 E-01	-9.05 E-01	-9.01 E-01	-9.01 E-01	N/A	-9.01 E-01
c_4	-1.40 E-05	-3.18 E-12	-5.00 E-06	-4.73 E-13	N/A	-5.00 E-06
c_5	-5.00 E-06	-1.12 E-11	-1.00 E-06	-4.68 E-14	N/A	-1.00 E-06
c_6	-7.03 E-01	-7.03 E-01	-7.03 E-01	-7.03 E-01	N/A	-7.03 E-01
c_7	-6.00 E-06	0	-3.00 E-06	0	N/A	-3.00 E-06
c_8	-5.83 E-01	-5.83 E-01	-5.83 E-01	-5.83 E-01	N/A	-5.83 E-01
c_9	-5.14 E-02	-5.13 E-02	-5.13 E-02	-5.13 E-02	N/A	-5.13 E-02
c_{10}	-6.00 E-06	0	-1.09 E-02	-1.09 E-02	N/A	-1.09 E-02
c_{11}	-1.98 E-01	-1.98 E-01	-1.98 E-01	-1.98 E-01	N/A	-1.98 E-01
$f(x)$	2994.499107	2994.471066	2996.356689	2996.348167	2994.471066	2994.471066

Table 4-3 Comparison of the Statistical Performance for the Speed-Reducer Problem

Method	Best (%)	Worst (%)	Mean (%)	CV (%)	FEs
HEA-ACT	9.36 E-04	9.39 E-03	4.75 E-03	2.34 E-05	40,000
DELC	0	0	0	6.35 E-16	30,000
MDE	6.30 E-02	6.41 E-02	6.33 E-02	2.74 E-06	24,000
PSO-DE	6.27 E-02	6.27 E-02	6.27 E-02	2.14 E-09	54,350
BSAISA	0	3.34 E-08	9.68 E-07	1.80 E-09	15,860
UAPSO ₁	0	1.46 E-05	1.07 E-06	2.80 E-08	15,000
UAPSO ₂	0	0	0	6.11 E-16	30,000

4.3.2 Pressure Vessel

Figure 4-6 Schematic Representation of the Pressure-Vessel Design Problem

The pressure-vessel design problem consists of a nonlinear objective function with three linear equality equations and one nonlinear inequality equation [102]. In this problem, a cylindrical pressure-vessel is capped at the two ends by hemispherical heads as displayed in Figure 4-6. The objective of this problem is to minimize the overall cost of manufacturing the pressure-vessel and

it consists of four design variables. Among the four design variables, the thickness of the shell T_s (x_1) and the thickness of the head T_h (x_2) are discrete variables that are integer multiples of 0.0625. The inner radius R (x_3) and the length of the cylindrical section of the vessel L (x_4) are continuous variables. Since this is a mixed integer problem, the UAPSO algorithm was modified to initially treat the discrete variables (x_1 and x_2) as continuous variables during the iterative process, but at the end of an iteration they were rounded off to the nearest integer value. This is a common approach for tackling discrete/integer variables [71, 103]. The pressure-vessel design problem is formulated according to the following equations [102]:

$$\min f(x) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3 \quad 4.1$$

$$c(x)_1 = -x_1 + 0.0193x_3 \leq 0 \quad 4.2$$

$$c(x)_2 = -x_2 + 0.00954x_3 \leq 0 \quad 4.3$$

$$c(x)_3 = \pi x_3^2 x_4 - \left(\frac{4\pi x_3^3}{3} \right) + 1.296(10^6) \leq 0 \quad 4.4$$

$$c(x)_4 = x_4 - 240 \leq 0 \quad 4.5$$

where the ranges of the design variables are formulated as follows:

$$0 \leq x_i \leq 100, \quad i = 1, 2$$

$$10 \leq x_i \leq 200, \quad i = 3, 4$$

The best solution obtained by UAPSO is $x = [0.8125, 0.4375, 42.0984, 176.6366]$ with $f(x) = 6059.7143$ in both cases of UAPSO. Comparison between the best solutions obtained by UAPSO with the other algorithms is presented in Table 4-4. The considered algorithms are the CPSO, CDE, GA-DT, BSA-SA_ε and BSAISA. The statistical performance of the considered methods with respect to the proposed method is presented in Table 4-5. As shown in Table 4-4, all considered methods satisfied the constraints applied in this problem. In terms of obtaining the best solution, BSA-SA_ε, BSAISA and UPASO managed to obtain the best solution,

$f(x) = 6059.7143$, while CPSO, CDE and GA-DT could not. BSA-SA $_{\epsilon}$ and BSAISA required 80,000 FEs and 16,320 FEs respectively, whereas UAPSO $_1$ required 15,000 only. As for the quality of results, UAPSO $_1$ performed the best compared to BSA-SA $_{\epsilon}$ and BSAISA since it produced a lower coefficient of variation and mean performance, however UAPSO $_2$ managed to perform better. Overall, in terms of convergence speed UAPSO $_1$ performed the best amongst the considered methods as it managed to obtain the best solution in the fewest FEs. On the other hand, in terms of robustness and effectiveness, UAPSO $_2$ performed the best across all statistical data.

Table 4-4 Comparison of the Best Solution for the Pressure-Vessel Problem

D. V.	CPSO	CDE	GA-DT	BSA-SA $_{\epsilon}$	BSAISA	UAPSO
x_1	8.13 E-01	8.13 E-01	8.13 E-01	8.13 E-01	8.13 E-01	8.13 E-01
x_2	4.38 E-01	4.38 E-01	4.38 E-01	4.38 E-01	4.38 E-01	4.38 E-01
x_3	4.21 E+01	4.21 E+01	4.21 E+01	4.21 E+01	4.21 E+01	4.21 E+01
x_4	1.77 E+02	1.77 E+02	1.77 E+02	1.77 E+02	1.77 E+02	1.77 E+02
c_1	-1.37 E-06	-6.67 E-07	-2.01 E-03	-9.50 E-10	0	-1.14 E-13
c_2	-3.59 E-04	-3.58 E-02	-3.58 E-02	-3.59 E-02	-3.59 E-02	-3.59 E-02
c_3	-1.19 E+02	-3.68 E+00	-2.49 E+01	-1.20 E-04	0	-1.21 E-08
c_4	-6.33 E+01	-6.34 E+01	-6.33 E+01	-6.34 E+01	-6.34 E+01	-6.34 E+01
$f(x)$	6061.0777	6059.7340	6059.9463	6059.7143	6059.7143	6059.7143

Table 4-5 Comparison of the Statistical Performance for the Pressure-Vessel Problem

Method	Best (%)	Worst (%)	Mean (%)	CV (%)	FEs
CPSO	2.25 E-02	5.02 E+00	1.44 E+00	1.41 E-02	30,000
CDE	3.25 E-04	5.14 E+00	4.21 E-01	7.07 E-03	204,800
GA-DT	3.83 E-03	6.76 E+00	1.94 E+00	2.12 E-02	80,000
BSA-SA $_{\epsilon}$	0	9.42 E-01	2.42 E-01	2.80 E-03	80,000
BSAISA	0	1.88 E+01	5.92 E+00	4.74 E-02	16,320
UAPSO $_1$	0	5.53 E-01	4.94 E-02	1.42 E-03	15,000
UAPSO $_2$	0	1.49 E-05	1.65 E-06	1.82 E-08	30,000

4.3.3 Tension/Compression Spring

Figure 4-7 Schematic Representation of the Tension/Compression Spring Design Problem

The tension/compression spring design problem consists of a nonlinear objective function with four nonlinear inequality equations [104] and is illustrated in Figure 4-7. The objective of this problem is to minimize the structural weight of the coil spring and it consists of three continuous design variables. The design variables are the wire diameter d (x_1), mean coil diameter D (x_2) and the number of active coils N (x_3). Furthermore, the spring is subjected to constraints on minimum deflection, shear stress, surge frequency and the outer diameter of the spring. The tension/compression spring design problem is formulated according to the following equations [104]:

$$\min f(\mathbf{x}) = (x_3 + 2)x_2x_1^2 \quad 4.6$$

$$c(\mathbf{x})_1 = 1 - \frac{x_2^3x_3}{71785x_1^4} \leq 0 \quad 4.7$$

$$c(\mathbf{x})_2 = \frac{4x_2^2 - x_1x_2}{12566(x_2x_1^3 - x_1^4)} + \frac{1}{5108x_1^2} - 1 \leq 0 \quad 4.8$$

$$c(\mathbf{x})_3 = 1 - \frac{140.45x_1}{x_2^2x_3} \leq 0 \quad 4.9$$

$$c(\mathbf{x})_4 = \frac{x_2 + x_1}{1.5} - 1 \leq 0 \quad 4.10$$

where the ranges of the design variables are formulated as follows:

$$0.05 \leq x_1 \leq 2.00$$

$$0.25 \leq x_2 \leq 1.30$$

$$2.00 \leq x_3 \leq 15.00$$

The best solution obtained by UAPSO is $x = [0.051712, 0.357264, 11.257028]$ with $f(x) = 0.012665$ in both cases of UAPSO. Comparison between the best solutions obtained by UAPSO with the other algorithms is presented in Table 4-6. The considered algorithms are the CPSO, MDE, PSO-DE, CDE and BSAISA. The statistical performance of the considered methods with respect to the proposed method is presented in Table 4-7. As shown in Table 4-6, all considered methods satisfied the constraints applied in this problem. In terms of obtaining the best solution, MDE, PSO-DE, BSAISA and UPASO managed to obtain the best solution, $f(x) = 0.012665$, while CPSO and CDE could not. MDE, PSO-DE and BSAISA required 24,000 FEs, 24,950 FEs and 9,440 FEs respectively, whereas UAPSO₁ required 9,000 only. In terms of the quality of results, UAPSO₁ performed the worst compared to MDE, PSO-DE and BSAISA, however UAPSO₂ managed to perform better than UAPSO₁ and MDE and was equal in performance to BSAISA. Comparing the statistical performance of UAPSO₂ and BSAISA – UAPSO₂ produced a similar mean performance, lower worst performance and higher coefficient of variation. Overall, in terms of convergence speed UAPSO₁ performed the best amongst the considered methods as it managed to obtain the best solution in the fewest FEs. On the other hand, in terms of robustness and effectiveness, UAPSO₂ performed comparable to the other considered methods.

Table 4-6 Comparison of the Best Solutions for the Tension/Compression Spring Design Problem

D. V.	CPSO	MDE	PSO-DE	CDE	BSAISA	UAPSO
x_1	5.17 E-02	5.17 E-02	5.17 E-02	5.16 E-02	5.17 E-02	5.17 E-02
x_2	3.58 E-01	3.57 E-01	3.57 E-01	3.55 E-01	3.57 E-01	3.57 E-01
x_3	1.12 E+01	1.13 E+01	1.13 E+01	1.14 E+01	1.13 E+01	1.13 E+01
c_1	-8.45 E-04	0	-1.99 E-09	-3.90 E-05	-6.38 E-10	-7.95 E-08
c_2	-1.26 E-05	0	-3.93 E-09	-1.83 E-04	-1.53 E-09	-6.06 E-08
c_3	-4.05 E+00	-4.05 E+00	-4.05 E+00	-4.05 E+00	-4.05 E+00	-4.05 E+00
c_4	-7.27 E-01	-7.28 E-01	-7.28 E-01	-7.29 E-01	-7.28 E-01	-7.27 E-01
$f(x)$	0.012675	0.012665	0.012665	0.012670	0.012665	0.012665

Table 4-7 Comparison of the Statistical Performance for the Tension/Compression Spring Design Problem

Method	Best (%)	Worst (%)	Mean (%)	CV (%)	FES
CPSO	7.90 E-02	2.05 E+00	5.13 E-01	4.08 E-02	240,000
MDE	0	7.11 E-02	7.90 E-03	1.58 E-04	24,000
PSO-DE	0	0	0	9.47 E-07	24,950
CDE	3.95 E-02	9.87 E-01	3.00 E-01	2.13 E-03	240,000
BSAISA	0	2.37 E-02	7.90 E-03	3.87 E-05	9,440
UAPSO ₁	0	4.74 E-02	3.16 E-02	1.37 E-04	9,000
UAPSO ₂	0	1.58 E-02	7.90 E-03	5.33 E-05	24,000

4.3.4 Welded Beam

Figure 4-8 Schematic Representation of the Welded-Beam Design Problem [105]

The welded beam design problem consists of a nonlinear objective function with seven nonlinear inequality equations [105] and is displayed in Figure 4-9. The objective of this problem is to minimize the cost subjected to constraints applied on shear stress (τ), bending stress in the beam (σ), buckling load on the bar (P_c), end deflection of the beam (δ) and side constraints. The design variables included in this problem are the height of the weld h (x_1), the length of the beam l (x_2), the thickness of the beam t (x_3) and the breadth of the beam b (x_4). The tension/compression spring design problem is formulated according to the following equations [105]:

$$\min f(x) = 1.10471x_2x_1^2 + 0.04811x_3x_4(14.0 + x_2) \quad 4.11$$

$$c(x)_1 = \tau(x) - \tau_{max} \leq 0 \quad 4.12$$

$$c(x)_2 = \sigma(x) - \sigma_{max} \leq 0 \quad 4.13$$

$$c(x)_3 = x_1 - x_4 \leq 0 \quad 4.14$$

$$c(x)_4 = 0.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5 \leq 0 \quad 4.15$$

$$c(x)_5 = 0.125 - x_1 \leq 0 \quad 4.16$$

$$c(x)_6 = \delta(x) - \delta_{max} \leq 0 \quad 4.17$$

$$c(x)_7 = P - P_c(x) \leq 0 \quad 4.18$$

where the other parameters are defined as:

$$\begin{aligned} \tau(x) &= \sqrt{(\tau')^2 + 2\tau'\tau''\left(\frac{x_2}{2R}\right) + (\tau'')^2}, \quad \tau' = \frac{P}{\sqrt{2}x_1x_2}, \quad \tau'' = \frac{MR}{J} \\ M &= P\left(L + \frac{x_2}{2}\right), \quad R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2}, \quad J = \frac{2x_1x_2}{\sqrt{2}}\left(\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right) \\ \sigma(x) &= \frac{6PL}{x_3^2x_4}, \quad \delta(x) = \frac{4PL^3}{Ex_3^3x_4}, \quad P_c(x) = \frac{4.013\sqrt{EG\left(\frac{x_3^2x_4^6}{36}\right)}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right) \end{aligned}$$

Also, the values of the parameters are set as thus:

$$P = 6000 \text{ lb}, \quad L = 14 \text{ in}, \quad E = 30 \times 10^6 \text{ psi}, \quad G = 30 \times 10^6 \text{ psi}$$

$$\tau_{max} = 13600 \text{ psi}, \quad \sigma_{max} = 30000 \text{ psi}, \quad \delta_{max} = 0.25 \text{ in}$$

where the ranges of the design variables are formulated as follows:

$$0.1 \leq x_i \leq 2, \quad i = 1,4$$

$$0.1 \leq x_i \leq 10, \quad i = 2,3$$

The best solution obtained by UAPSO is $x = [0.205730, 3.470489, 9.036624, 0.205730]$ with $f(x) = 1.724852$ in both cases of UAPSO. Comparison between the best solution obtained by UAPSO with the other considered algorithms is presented in Table 4-8. The considered algorithms are the CPSO, HPSO, PSO-DE, MBA and BSAISA. The statistical performance of the considered methods with respect to the proposed method is presented in Table 4-9. As shown in Table 4-8, all considered methods satisfied the constraints applied in this problem. In terms of obtaining the best solution, HPSO, PSO-DE and BSAISA managed to obtain the best solution, $f(x) = 1.724852$, while CPSO and MBA could not. HPSO, PSO-DE and BSAISA required 81,000 FEs, 66,600 FEs and 29,000 FEs respectively, whereas UAPSO₁ required 20,000 only. In terms of the quality of results, UAPSO₁ performed better than HPSO but worse than PSO-DE and BSAISA, however UAPSO₂ managed to perform better than UAPSO₁, HPSO and BSAISA. Overall, in terms of convergence speed UAPSO₁ performed the best amongst the considered methods as it managed to obtain the best solution in the fewest FEs. On the other hand, in terms of robustness and effectiveness, UAPSO₂ performed the second best, across all statistical data, after PSO-DE.

Table 4-8 Comparison of the Best Solutions for the Welded Beam Design Problem

D. V.	CPSO	HPSO	PSO-DE	MBA	BSAISA	UAPSO
x_1	2.02 E-01	2.06 E-01	2.06 E-01	2.06 E-01	2.06 E-01	2.06 E-01
x_2	3.54 E+00	3.47 E+00	3.47 E+00	3.47 E+00	3.47 E+00	3.47 E+00
x_3	9.05 E+00	9.04 E+00	9.04 E+00	9.04 E+00	9.04 E+00	9.04 E+00
x_4	2.06 E-01	2.06 E-01	2.06 E-01	2.06 E-01	2.06 E-01	2.06 E-01
c_1	-1.28 E+01	-2.54 E-02	-1.51 E-05	-1.61E-03	0	-5.64 E-05
c_2	-1.25 E+00	-5.31 E-02	-2.88 E-05	-1.69E-02	0	-7.91 E-05
c_3	-1.49 E-03	0	0	-2.40 E-07	-5.55 E-12	-8.52 E-12
c_4	-3.43 E+00	-3.43 E+00	-3.43 E+00	-3.43 E+00	-3.43 E+00	-3.43 E+00
c_5	-7.94 E-02	-8.07 E-02	-8.07 E-02	-8.07 E-02	-8.07 E-02	-8.07 E-02
c_6	-2.36 E-01	-2.36 E-01	-2.36 E-01	-2.36 E-01	-2.36 E-01	-2.36 E-01
c_7	-1.17 E+01	-3.16 E-02	-1.86 E-05	-1.46 E-03	-5.46 E-12	-1.40 E-05
$f(x)$	1.728024	1.724852	1.724852	1.724853	1.724852	1.724852

Table 4-9 Comparison of the Statistical Performance for the Welded Beam Design Problem

Method	Best (%)	Worst (%)	Mean (%)	CV (%)	FES
CPSO	1.84 E-01	3.32 E+00	1.39 E+00	7.38 E-03	240,000
HPSO	0	5.19 E+00	1.40 E+00	2.29 E-02	81,000
PSO-DE	0	0	0	3.88 E-16	66,600
MBA	5.80 E-05	5.80 E-05	5.80 E-05	4.02 E-19	47,340
BSAISA	0	1.16 E-04	0	1.36 E-07	29,000
UAPSO ₁	0	3.83 E-03	1.45 E-03	1.11 E-05	20,000
UAPSO ₂	0	5.80 E-05	5.80 E-05	9.39 E-08	50,000

4.4 Summary

In this section, a method called the Unique Adaptive Particle Swarm Optimization (UAPSO) was proposed to self-adapt the particle swarm optimization. The proposed method utilizes a feedback system that is controlled by the evolutionary state of a particle. Evolutionary state is a novel term that measures the performance of a particle's latest feasible solution with its historical best solution and the best solution of the swarm. By using the feedback system, the proposed method adapts both the inertia weight and the acceleration coefficients (through the observation of the environment) by modifying them to help promote a more effective and efficient search.

Four constrained optimization problem were considered to validate the performance of the proposed method. The numerical results from the comparative study demonstrated the effectiveness of UAPSO in handling and solving different types of constrained optimization problems. The obtained results illustrated that the proposed method managed to obtain the best global solution in the fewest function evaluations compared to the other considered methods; this indicates the superiority of UAPSO's convergence speed. Furthermore, UAPSO managed to perform robustly and efficiently in terms of its statistical performance by consistently producing solutions close to the global solution across the 100 independent runs. The proposed method succeeded in obtaining very satisfactory results and in cases where it did not manage to perform better than the considered methods, it produced competitive results. This proves that UAPSO may be used to solve practical engineering optimization problems. Finally, the computational cost and quality of results derived by UAPSO depends on the nature of the problem and this is true for any metaheuristic algorithms.

Chapter 5: Application of the Proposed Method in Solving a Practical Structural Optimization Problem

In this chapter, the application of the proposed method in solving a real-world structural optimization problem is studied. The real-world problem involves designing a three-bar truss structure. In theory, any structure that is cheap to produce and reliable is desirable which is why the weight minimization of structural components is highly sought-after. The considered problem is nonlinear and subjected to performance and technological related constraints such as member crushing, member buckling, size limitation, spatial distance limitation, failure by excessive displacement and failure by resonance when the natural frequency is lower than a certain threshold. Different forms of the three-bar truss problem have been studied previously in literature [104, 106, 107].

The three-bar truss design problem is statically indeterminate and thus cannot be solved for by simply using the equilibrium equations. The structure is designed, for minimization of weight to support a load, using the cross-sectional area of each bar and its location. This problem is conflicting since if it was required to reduce the stress and displacement that a system supports, the cross-sectional area is simply increased however, this would result in an increase in the structural weight. Furthermore, the location of the bar influences the stress, displacement, buckling and resonance. Due to the nature of this optimization problem, it was selected to test the capability of the proposed method, the UAPSO algorithm. Hence, this problem allows exploring the effectiveness of UAPSO in terms of robustness and efficiency. The performance of UAPSO, by means of statistical performance, is compared with five other variants of the PSO method including the original PSO version and the Sequential Quadratic Programming (SQP) method, a derivative based search method.

Figure 5-1 Schematic Diagram of the Three-Bar Truss Design Problem

5.1 Optimization Problem

The objective of this optimization problem is the minimization of the structural weight of a three-bar structure, shown in the Figure 5-1. The design variables are the cross-sectional areas a_1, a_2, a_3 and the coordinates x_1, x_2, x_3 of nodes A, B, C. The members are made from the same material with an allowable strength S_a , Young modulus E and density ρ . The structure is subjected to load P at angle θ which results in horizontal displacement u and vertical displacement v . The system parameters are presented in Table 5-1. The three-bar truss design problem is modeled into an optimization problem as follows:

$$\min f(x) = W = \rho g \sum_{i=1}^3 a_i \sqrt{x_i^2 + 0.5^2} \quad 5.1$$

$$c(x)_1 = \left| \frac{E(u x_i + 0.5 v_i)}{x_i^2 + 0.5^2} \right| - S_a \leq 0 \quad \text{for } i = 1, 2, 3 \quad 5.2$$

$$c(x)_2 = E a_i \frac{u x_i + 0.5 v}{\sqrt{x_i^2 + 0.5^2}} - \frac{\pi E a_i}{4(x_i^2 + 0.5^2)} \leq 0 \quad \text{for } i = 1, 2, 3 \quad 5.3$$

$$c(x)_3 = -a_i + 3.2 \times 10^{-6} \leq 0 \quad \text{for } i = 1, 2, 3 \quad 5.4$$

$$c(x)_4 = x_1 - x_2 + 0.1 \leq 0 \quad 5.6$$

$$c(x)_5 = x_2 - x_3 + 0.1 \leq 0 \quad 5.7$$

$$c(x)_6 = \left| \sqrt{u^2 + v^2} \right| \leq \delta \quad 5.8$$

$$c(x)_7 = -\min(\omega) + \omega_0 \leq 0 \quad 5.9$$

where the ranges of the design variables are formulated as follows:

$$-1 \leq x_i \leq 1, \quad i = 1, 2, 3$$

$$-1 \leq a_i \leq 1, \quad i = 1, 2, 3$$

Due to the statically indeterminate state of the problem, the analytical technique of finite element method was applied to derive the mathematical expressions of the optimization problem presented above. The displacements u and v are calculated by the following equation:

$$E \sum_{i=1}^3 \frac{a_i}{(x_i^2 + 0.5^2)^{3/2}} \begin{bmatrix} x_i^2 & 0.5 x_i \\ 0.5 x_i & 0.5^2 \end{bmatrix} \begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{Bmatrix} P \cos \theta \\ -P \sin \theta \end{Bmatrix} \quad 5.10$$

The above equations have been derived from the general equation of a 2D truss element given by the following equation [108]:

$$\frac{EA_i}{L_i} \begin{bmatrix} \cos^2 \theta_i & \cos \theta_i \sin \theta_i & -\cos^2 \theta_i & -\cos \theta_i \sin \theta_i \\ \cos \theta_i \sin \theta_i & \sin^2 \theta_i & -\cos \theta_i \sin \theta_i & -\sin^2 \theta_i \\ -\cos^2 \theta_i & -\cos \theta_i \sin \theta_i & \cos^2 \theta_i & \cos \theta_i \sin \theta_i \\ -\cos \theta_i \sin \theta_i & -\sin^2 \theta_i & \cos \theta_i \sin \theta_i & \sin^2 \theta_i \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \end{Bmatrix} = \begin{Bmatrix} P_{ix} \\ P_{iy} \\ P_{jx} \\ P_{jy} \end{Bmatrix} \quad 5.11$$

where u refers to the horizontal displacement of elements i and j , v refers to the vertical displacement of elements i and j and P refers to the applied load along directions x and y on elements i and j .

The phenomenon of resonance is undesirable, since it could lead to system failure [109]. Therefore, the three-bar truss is designed in such a way that its lowest excited natural frequency exceeds a certain threshold. The system has two degrees of freedom. Hence, there are two natural frequencies. After deriving the mass and stiffness matrices, the natural frequency of the system can be calculated by solving the following characteristic equation:

$$\mathbf{K} - \mathbf{M}(\omega^2) = 0 \quad 5.12$$

where $\mathbf{K}$ is the stiffness matrix, $\mathbf{M}$ the mass matrix and ω is the frequency vector. By solving the above, the two frequencies are derived and the lower frequency is subjected to a constraint.

Table 5-1 System Parameters of the Three-Bar Truss Design Problem

System Parameter	Parameter Value
Length (L)	0.5 m
Load (P)	70 kN
Angle of Applied Load (θ)	30°
Density (ρ)	7800 kg/m^3
Gravity (g)	9.81 m/s^2
Allowable Strength (S_a)	250 MPa
Young's Modulus (E)	200 GPa
Maximum Relative Displacement (δ)	10 mm
Natural Frequency Threshold (ω_0)	100 Hz

5.2 Optimization Problem Setup

The proposed method is applied to the three-bar truss design problem along with the original PSO [1], PSO-LDIW [50], PSO-NLIW [54], PSO-RIW [52], PSO-CIW [55] and the SQP method [5]. The SQP method is a widely used derivative-based method that utilizes second order information to find the optimal solution. Since SQP is a deterministic method, i.e. the algorithm solves the problem without any stochastic behaviour, statistical data cannot be generated and therefore the optimal solution will only be used in the comparison. The design problem was

formulated with a swarm size of 10 and the maximum number of iterations was set as 2000, whereas SQP was initiated, employed through the MATLAB function *fmincon*, with a starting design of $[-1, -1, 1, 3.2e - 06, 3.2e - 06, 3.2e - 06]$. The maximum velocity was set to the dynamic range of the design variables. The dynamic range of the considered inertia weight variants are listed in Table 5-2. Similarly, here the problem was run 100 times to extract statistical performance in terms of best function value, worst function value, mean function value and standard deviation. The MATLAB code of the Three-Bar Truss Design Problem is provided in the Appendices.

Table 5-2 Dynamic Range of the Considered PSO Variants

PSO Variant	Inertia Weight Range
PSO-LDIW [50]	0.4 – 0.9
PSO-NLIW [54]	0.4 – 0.9
PSO-RIW [52]	0.5 – 1.0
PSO-CIW [55]	0.0 – 1.0

5.3 Numerical Results

The best solution obtained by UAPSO is $x = [-0.971594, -0.867017, -0.710397, 3.20 \text{ E} - 06, 2.74 \text{ E} - 04, 3.20 \text{ E} - 06]$ with $f(x) = 21.440613$. Comparison between the best solutions obtained by UAPSO, the other considered PSO variants and SQP is presented in Table 5-3. The statistical performance of the considered PSO variants with respect to the proposed method is presented in Figure 5-2. The error bar represents the distribution of results as per the standard deviation. In terms of obtaining the best solution, UAPSO managed to obtain the best solution $f(x) = 21.440613$ among the considered techniques. As for the quality of results, UAPSO performed the best since its statistical performance was superior compared to the rest; UAPSO's mean function value, worst function value and standard deviation are all lower than the

other techniques. Table 5-4 contains the performance of the considered methods in terms of satisfying the applied constraints. SQP, PSO-LDIW and PSO-NLIW failed to satisfy all applied constraints and thus failed to solve the design problem successfully.

Table 5-3 Comparison of the Best Solutions for the Three-Bar Truss Design Problem

D. V.	SQP	PSO	PSO-LDIW	PSO-NLIW	PSO-RIW	PSO-CIW	UAPSO
x_1	-9.35 E-01	-1.00 E+00	-1.00E+00	-1.00 E+00	-1.00 E+00	-1.00 E+00	-9.72 E-01
x_2	-8.35 E-01	-8.70 E-01	-8.67E-01	-8.67 E-01	-8.67 E-01	-8.67 E-01	-8.67 E-01
x_3	-2.99 E-01	-4.43 E-01	-7.18E-01	-7.18 E-01	-7.18 E-01	-7.41 E-01	-7.10 E-01
x_4	9.32 E-05	3.20 E-06	3.20 E-06	3.20 E-06	3.20 E-06	3.20 E-06	3.20 E-06
x_5	1.87 E-04	2.76 E-04	2.74 E-04	2.72 E-04	2.74 E-04	2.74 E-04	2.74 E-04
x_6	2.02 E-06	3.20 E-06	3.22 E-06	4.87 E-06	3.20 E-06	3.20 E-06	3.20 E-06
$f(x)$	21.5615	21.6408	21.4432	21.4432	21.4432	21.4429	21.4406

Figure 5-2 Statistical Performance of UAPSO and the Considered PSO Variants in the Three-Bar Truss Problem

Table 5-4 Sum of Violated Constraints by the Considered PSO Variants and SQP

Method	Sum of Violated Constraints
SQP	2.24 E+04
PSO	0
PSO-LDIW	6.42 E-05
PSO-NLIW	6.91 E-05
PSO-RIW	0
PSO-CIW	0
UAPSO	0

5.3.1 Comparison of Convergence Performance

The statistical performance of UAPSO shows its superiority compared to the other variants however in order to determine whether UAPSO managed to avoid premature convergence, the convergence curves of all considered variants, including UAPSO, are studied in Figure 5-2. The convergence curves display the historical performance of each variant's best obtained solution throughout the 100 runs. From Figure 5-2, it can be determined that the six variants performed in a similar manner by converging in the last 1000 iterations. The robustness of an optimization method is revealed by its ability in avoiding premature convergence and this can be seen in the zoomed-in section of Figure 5-2. The zoomed-in section shows that the variants have, all, gone through multiple phases where particles were stuck in local minima and that apart from UAPSO, the other variants converged to their final solution in the final 500 iterations. In contrary, UAPSO kept on discovering better solutions until converging in the final 50 iterations and this is despite of UAPSO's slower start, compared to the other methods, as evidenced by its poorer performance in the initial 1000 iterations. This shows the robustness of the proposed method and coupled with its strong statistical performance, the UAPSO algorithm proved to be successful in navigating the particles through the search space compared to the other variants.

Figure 5-3 Convergence Curves of UAPSO and the Considered Variants for the Three-Bar Truss Design Problem

5.3.2 Performance of UAPSO with Varying Function Evaluations

UAPSO managed to outperform the considered variants with an equal number of function evaluations. Figure 5-4 shows the statistical performance of UAPSO with varying function evaluations. The number of function evaluations was reduced by 50% from 20,000 FEs to 10,000 FEs. Here, UAPSO was run 100 times to extract statistical data. Naturally, the performance of UAPSO was affected negatively with the reduction of the function evaluations but comparing the performance of UAPSO at lower FEs with the considered PSO variants reveals the effectiveness of UAPSO. The best performing PSO variant (excluding UAPSO) was PSO-CIW; UAPSO with 20% less FEs managed to perform better across all statistical measures as can be seen in Table 5-5.

Figure 5-4 Statistical Performance of UAPSO with Varying Function Evaluations

Table 5-5 Comparison between PSO-CIW with 20,000 FEs and UAPSO with 16,000 FEs

Variant	Best	Worst	Mean	Standard Deviation
PSO-CIW (with 20,000 FEs)	21.4429	25.2582	22.2640	1.00
UAPSO (with 16,000 FEs)	21.4426	21.7407	21.5867	0.11

The consistent and efficient behaviour of UAPSO was demonstrated at lower FEs as UAPSO managed to produce a lower worst and mean value, even with 50% less FEs, in comparison with the considered variants as can be seen in Figure 5-5. The results show that on average UAPSO with 50% less FEs produced a three-bar truss that weighed between 7% to 12% less in structural weight in comparison with the considered PSO variants.

Figure 5-5 Comparison of the Mean Function Value and Distribution of Results Produced by UAPSO (with 50% less FEs) with the Considered PSO Variants.

5.4 Summary

In this section, the application of the proposed method was studied in solving a real-world engineering design problem. Optimal design of a three-bar truss was the chosen engineering problem. The proposed method was applied to the design problem to obtain the most optimal geometric properties with the objective of minimizing the structural weight, while being subjected to performance and technological related constraints. Numerical results proved that the proposed method managed to outperform the considered PSO variants and the SQP method in obtaining the best solution and in its overall statistical performance.

In addition, PSO-LDIW, PSO-NLIW and SQP failed to satisfy all applied constraints, whereas UAPSO and the remaining variants were successful. Furthermore, the results displayed the ability of the proposed method in avoiding premature convergence contrary to the other PSO variants. Lastly, studying the performance of UAPSO at lower numbers of function evaluations revealed that it was able to outperform the considered methods by producing a more consistent statistical performance.

Chapter 6: Conclusion

The Particle Swarm Optimization (PSO) algorithm is a metaheuristic method that takes its inspiration from the foraging behavior of groups of animals such as a flock of birds and a school of fish. PSO belongs to the class of nature – inspired search methods and could also be considered as direct search method since it does not make use of any higher-order information to solve the optimization problem. This behavior makes PSO and the other nature-inspired methods more desirable over the other methods. PSO's ability to solve problems in fewer evaluations with a high efficiency sets it apart from the other metaheuristic methods. However, PSO, in its original form, is not without its flaws. PSO's major flaw is falling into local solutions and being stuck there, leading to premature convergence. Therefore, researchers tried coming up with various ways to improve the algorithm. One such improvement was the development of the inertia weight and in subsequent developments, the inertia weight has gone through different dynamic forms. Another flaw of the original form of the PSO algorithm is its sensitivity to the control settings. Due to the two aforementioned flaws, many variants of PSO have been developed either by merging concepts of PSO with other metaheuristic methods or by introducing novel concepts to the PSO algorithm. One such class of PSO variants attempts to self-adapt the algorithm through a feedback system – this class is known as the Adaptive Particle Swarm Optimization (APSO) methods. In order for any modified PSO variant to handle constraints, a scheme needs to be implemented such that the particles can learn from each other's feasible solutions only. Therefore, the constraint handling technique should work in accord with the feedback system to ensure that the particles are adapting to the environment by directing itself to the feasible regions only.

6.1 Contributions and Significance

In this thesis, a novel scheme of control parameter adaption was developed. The proposed method, known as the Unique Adaptive Particle Swarm Optimization (UAPSO), was developed to improve convergence speed and robustness of the algorithm. The following lists the major contributions of this research work:

- An extensive parametric sensitivity analysis was performed to study the impact of the individual control parameters in the original PSO algorithm. The parametric study conducted considered a constrained problem, which is according to the best knowledge of the author the first foray into sensitivity analysis of the PSO algorithm with respect to a constrained environment. The study revealed that the inertia weight and acceleration coefficients have the most significant effect on the performance of the algorithm and that optimal tuning of the inertia weight and acceleration coefficients would overcome the need to initiate the algorithm with a large swarm size and would thus reduce the computational effort.
- A self-adapting PSO algorithm was developed that makes use of a feedback system to update the inertia weight and the acceleration coefficients. In the developed method, only feasible solutions are retained by the particles for learning. A feedback parameter known as the evolutionary state was introduced which is a measure of the current performance of the particles with respect to the best historical solutions. Through the evolutionary state, a unique inertia weight and unique acceleration coefficients are established for each particle. Furthermore, a reinitialization scheme was implemented to avoid stagnation of momentum.

- The performance of UAPSO was studied by solving four benchmark constrained engineering design problems. The considered problems were the Speed-Reducer, Pressure Vessel, Tension/Compression Spring and Welded Beam design problems. The numerical results obtained by UAPSO were compared to some well-known algorithms by means of statistical performance. The comparative study demonstrated that UAPSO proved to be effective and efficient in solving the considered problems and especially in terms of the speed of convergence. The study illustrated that UAPSO managed to find the best solution at the fewest evaluations compared to the other algorithms in all four problems. In terms of the quality of results, UAPSO managed to perform robustly and outperformed the majority of the considered algorithms.
- A real-world structural optimization problem was considered to test the capability of the proposed method in solving a highly nonlinear problem. UAPSO managed to successfully solve the problem and obtain the optimal geometric parameters. Moreover, UAPSO outperformed the other considered PSO variants by solving the problem more efficiently and robustly. In addition, the results of the numerical simulation demonstrated the ability of UAPSO in avoiding premature convergence. Lastly, the consistency and efficiency of UAPSO was demonstrated by studying its statistical performance at varying number of function evaluations.

6.2 Recommendations for Future Work

The overall results of the research work proved successful, however listed below are some suggestions for further work in this line of research:

- A more comprehensive parametric sensitivity analysis would help glean more information over the impact of the parameters. In this research work, each parameter was studied individually and this helped in understanding the performance of PSO with respect to the individual parameter however by studying a combination of different parameters simultaneously, it might lead to more insight on the PSO algorithm.
- The developed method is both easily computable and effective. The obtained results prove the potential of the algorithm and in particular its superior convergence speed. However, in some cases the proposed method slightly underperformed when compared to other well-known methods. By incorporating other optimization algorithms, i.e. hybridization, the robustness of UAPSO could improve and lead to a more efficient performance.
- One of the major concerns when formulating the proposed method was related to computational cost however, this could have been neglected by making use of parallel computing. Parallel computing would allow the algorithm to perform multiple calculations simultaneously and thus reduce the run time. Introducing a more complex and demanding change to the PSO algorithm would be possible by utilizing parallel computing. This might also improve the robustness and efficiency of the PSO algorithm.

References

- [1] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *IEEE International Conference on Neural Networks*, Perth, Australia, 1995.
- [2] J. S. Arora, *Introduction to Optimum Design*, 4th ed., Elsevier, Ed., Academic Press, 2017.
- [3] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., Springer New York, 2006.
- [4] M. Dorigo and T. Stutzle, "Ant Colony Optimization: Overview and Recent Advances," in *Handbook of Metaheuristics*, Boston, MA, Springer, 2010, pp. 227-263.
- [5] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming for large-scale nonlinear optimization," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1-2, pp. 123-137, 2000.
- [6] M. J. D. Powell, "A Fast Algorithm for Nonlinearly Constrained Optimization Calculations," in *Numerical Analysis. Lecture Notes in Mathematics*, 1978.
- [7] R. M. Lewis, V. Torczon and M. W. Trosset, "Direct Search Methods: Then and Now," *Journal of Computational and Applied Mathematics*, vol. 124, p. 191–207, 2000.
- [8] M. W. Trosset, "I Know It When I See It: Toward a Denition of Direct Search Methods," *SIAG/OPT Views-and-News*, vol. 9, pp. 7-10 , 1997.
- [9] R. Hooke and T. A. Jeeves, ""Direct Search" Solution of Numerical and Statistical Problems," *Journal of the ACM*, vol. 8, no. 2, pp. 212-229, 1961 .
- [10] W. H. Swann, "Direct Search Methods," in *Numerical Methods for Unconstrained Optimization*, New York, Academic Press, 1972, p. 13–28.

- [11] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *The Computer Journal*, vol. 7, no. 4, p. 308–313, 1965.
- [12] G. E. P. Box and K. B. Wilson, "On the Experimental Attainment of Optimum Conditions," *Journal of the Royal Statistical Society*, vol. 13, no. 1, pp. 1-45, 1951.
- [13] G. E. P. Box and D. W. Behnken, "Some New Three Level Designs for the Study of Quantitative Variables," *Technometrics*, vol. 2, no. 4, pp. 455-475, 1960.
- [14] W. J. Roux, N. Stander and R. T. Haftka, "Response Surface Approximations for Structural Optimization," *International Journal for Numerical Methods in Engineering*, vol. 42, p. 517–534, 1998.
- [15] X. S. Yang, *Nature-Inspired Optimization Algorithms*, London: Elsevier, 2014.
- [16] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Cambridge, MA: MIT Press, 1975.
- [17] M. Srinivas and L. M. Patnaik, "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 4, pp. 656-667, 1994.
- [18] M. Dorigo, V. Maniezzo and A. Coloni, "The Ant System: Optimization by a colony of cooperating agents," *IEEE transactions on systems, man and cybernetics. Part B, Cybernetics*, vol. 26, no. 1, pp. 29-41, 1996.
- [19] M. Dorigo, "Optimization, Learning and Natural Algorithms. Ph.D.Thesis," Politecnico di Milano, Italy, 1992.

- [20] G. Bilchev and I. C. Parmee, "The Ant Colony Metaphor for Searching Continuous Design Spaces," in *Evolutionary Computing. AISB EC 1995. Lecture Notes in Computer Science, vol 993.*, Springer, Berlin, Heidelberg, 1995.
- [21] M. Mathur, S. B. Karale, S. Priye, V. K. Jayaraman and B. D. Kulkarni, "Ant Colony Approach to Continuous Function Optimization," *Industrial & Engineering Chemistry Research*, vol. 39, no. 10, pp. 3814-3822, 2000 .
- [22] A. P. Engelbrecht, "Particle Swarm Optimization," in *Computational Intelligence: An Introduction*, Hoboken, NJ, John Wiley & Sons, 2007, pp. 289-358.
- [23] R. C. Eberhart and Y. Shi, "Comparison between Genetic Algorithms and Particle Swarm Optimization," in *Evolutionary Programming VII. EP 1998. Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 1998.
- [24] P. J. Angeline, "Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences," in *Evolutionary Programming VII. EP 1998. Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 1998.
- [25] X. Hu, R. C. Eberhart and Y. Shi, "Engineering Optimization with Particle Swarm," in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, Indianapolis, IN, 2003.
- [26] S. Panda and N. P. Padhy, "Comparison of Particle Swarm Optimization and Genetic Algorithm for FACTS-based Controller Design," *Applied Soft Computing*, vol. 8, no. 4, pp. 1418-1427, 2008.
- [27] M. A. Panduro, C. A. Brizuela, L. I. Balderas and D. A. Acosta, "A Comparison of Genetic Algorithms, Particle Swarm Optimization and the Differential Evolution Method for the

- Design of Scannable Circular Antenna Arrays," *Progress In Electromagnetics Research B*, vol. 13, p. 171–186, 2009.
- [28] V. G. Gudise and G. K. Venayagamoorthy, "Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks," in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, Indianapolis, IN, 2003.
- [29] Y. Wang, J. Lv, L. Zhu and Y. Ma, "Crystal Structure Prediction via Particle Swarm Optimization," *Physical Review B*, vol. 82, no. 9, pp. 094116-1 - 094116-8, 2010.
- [30] S. Pandey, L. Wu, S. M. Guru and R. Buyya, "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments," in *24th IEEE International Conference on Advanced Information Networking and Applications*, Perth, WA, 2010.
- [31] A. Carlisle and G. Dozier, "An Off-The-Shelf PSO," in *Proceeding of Workshop on Particle Swarm Optimization*, Indianapolis, USA, 2001.
- [32] F. van den Bergh and A. Engelbrecht , "A New Locally Convergent Particle Swarm Optimiser," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, Yasmine Hammamet, Tunisia, 2002.
- [33] R. Eberhart and J. Kennedy, "A New Optimizer Using Particle Swarm Theory," in *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Nagoya, Japan, 1995.
- [34] M. R. Bonyadi and Z. Michalewicz, "Particle Swarm Optimization for Single Objective Continuous Space Problems: A Review," *Evolutionary Computation*, vol. 25, no. 1, pp. 1-54, 2017 .

- [35] J. Kennedy, "Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance," in *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, Washington, DC, 1999.
- [36] R. Mendes, J. Kennedy and J. Neves, "The Fully Informed Particle Swarm: Simpler, Maybe Better," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 204 - 210, 2004.
- [37] J. Kennedy and R. Mendes, "Population Structure and Particle Swarm Performance," in *Proceedings of the 2002 Congress on Evolutionary Computation*, Honolulu, HI, 2002.
- [38] J. Kennedy and R. Mendes, "Neighborhood Topologies in Fully Informed and Best-Of-Neighborhood Particle Swarms," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 36, no. 4, pp. 515 - 519, 2006.
- [39] R. Poli, J. Kennedy and T. Blackwell, "Particle Swarm Optimisation: An Overview," *Swarm Intelligence*, vol. 1, no. 1, p. 33–57, 2007.
- [40] P. N. Suganthan, "Particle Swarm Optimiser with Neighbourhood Operator," in *Proceedings of the 1999 Congress on Evolutionary Computation*, Washington, DC, 1991.
- [41] A. Ratnaweera, S. K. Halgamuge and H. C. Watson, "Self-Organizing Hierarchical Particle Swarm Optimizer With Time-Varying Acceleration Coefficients," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 240 - 255, 2004 .
- [42] Y. Shi and R. C. Eberhart, "Parameter Selection in Particle Swarm Optimization," in *Evolutionary Programming VII. EP 1998. Lecture Notes in Computer Science*, Springer, Berlin, 1998.
- [43] F. Van den Bergh and A. Engelbrecht, "A Study of Particle Swarm Optimization Particle Trajectories," *Information Sciences*, vol. 176, no. 8, pp. 937-971, 2006.

- [44] A. P. Engelbrecht, *Computational Intelligence: An Introduction*, 2nd Edition, Wiley, 2007.
- [45] R. C. Eberhart and Y. Shi, "Particle Swarm Optimization: Developments, Applications and Resources," in *Proceedings of the 2001 Congress on Evolutionary Computation*, Seoul, 2001.
- [46] A. Banks, J. Vincent and C. Anyakoha, "A Review of Particle Swarm Optimization. Part I: Background and Development," *Natural Computing*, vol. 6, no. 4, p. 467–484, 2007.
- [47] L. dos Santos Coelho, "Gaussian Quantum-Behaved Particle Swarm Optimization Approaches for Constrained Engineering Design Problems," *Expert Systems with Applications*, vol. 37, p. 1676–1683, 2010.
- [48] I. C. Trelea, "The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection," *Information Processing Letters*, vol. 85, no. 6, pp. 317-325, 2003.
- [49] F. van den Bergh and A. P. Engelbrecht, "A Cooperative Approach to Particle Swarm Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225 - 239, 2004 .
- [50] Y. Shi and R. Eberhart, "A Modified Particle Swarm Optimizer," in *IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence*, Anchorage, USA, 1998.
- [51] R. C. Eberhart and Y. Shi, "Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization," in *Proceedings of the 2000 Congress on Evolutionary Computation*, La Jolla, CA, 2000.
- [52] R. C. Eberhart and Y. Shi, "Tracking and Optimizing Dynamic Systems with Particle Swarms," in *Proceedings of the 2001 Congress on Evolutionary Computation*, Seoul, 2001.

- [53] Y. Shi and R. C. Eberhart, "Fuzzy Adaptive Particle Swarm Optimization," in *Proceedings of the 2001 Congress on Evolutionary Computation*, Seoul, South Korea, 2001.
- [54] A. Chatterjee and P. Siarry, "Nonlinear Inertia Weight Variation for Dynamic Adaptation in Particle Swarm Optimization," *Computers & Operations Research*, vol. 33, no. 3, p. 859–871, 2006.
- [55] Y. Feng, G.-F. Teng, A.-X. Wang and Y.-M. Yao, "Chaotic Inertia Weight in Particle Swarm Optimization," in *Second International Conference on Innovative Computing, Informatio and Control*, Kumamoto, 2007.
- [56] M. Taherkhani and R. Safabakhsh, "A Novel Stability-based Adaptive Inertia Weight for Particle Swarm Optimization," *Applied Soft Computing*, vol. 38, pp. 281-295, 2016.
- [57] E. Mezura-Montes and C. A. Coello Coello, "Constraint-Handling in Nature-Inspired Numerical Optimization: Past, Present and Future," *Swarm and Evolutionary Computation*, vol. 1, no. 4, pp. 173-194, 2011.
- [58] C. A. Coello Coello, "Theoretical and Numerical Constraint-Handling Techniques Used with Evolutionary Algorithms: A Survey of the State of the Art," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11-12, p. 1245–1287, 2002.
- [59] X. Hu and R. Eberhart, "Solving Constrained Nonlinear Optimization Problems," in *Proceedings of the Sixth World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, Florida, 2002.
- [60] E. Mezura-Montes and C. A. Coello Coello, "An Empirical Study About the Usefulness of Evolution Strategies to Solve Constrained Optimization Problems," *International Journal of General Systems*, vol. 37, no. 4, pp. 443-473, 2008.

- [61] Z. Michalewicz and M. Schoenauer, "Evolutionary Algorithms for Constrained Parameter Optimization Problems," *Evolutionary Computation*, vol. 4, no. 1, p. 1–32, 1996.
- [62] J. T. Richardson, M. R. Palmer, G. E. Liepins and M. Hilliard, "Some Guidelines for Genetic Algorithms with Penalty Functions," in *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, California, 1989.
- [63] J. Kennedy, "The Particle Swarm: Social Adaptation of Knowledge," in *Proceedings of 1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, IN, 1997.
- [64] M. Clerc and J. Kennedy, "The Particle Swarm - Explosion, Stability, and Convergence in a Multidimensional Complex Space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58 - 73, 2002.
- [65] Q. He and L. Wang, "An Effective Co-Evolutionary Particle Swarm Optimization for Constrained Engineering Design Problems," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 1, p. 89–99, 2007.
- [66] C.-F. Juang, "A Hybrid of Genetic Algorithm and Particle Swarm Optimization for Recurrent Network Design," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 3, no. 2, p. 997–1006, 2004.
- [67] Y. Jiang, T. Hu, C. C. Huang and X. Wu, "An Improved Particle swarm Optimization Algorithm," *Applied Mathematics and Computation*, vol. 193, no. 1, p. 231–239, 2007.
- [68] P. S. Shelokar, P. Siarry, V. K. Jayaraman and B. D. Kulkarni, "Particle Swarm and Ant Colony Algorithms Hybridized for Improved Continuous Optimization," *Applied Mathematics and Computation*, vol. 188, no. 1, p. 129–142, 2007.

- [69] A. Carlisle and G. Dozier, "Adapting Particle Swarm Optimization to Dynamic Environments," in *Proceedings of international conference on artificial intelligence*, Las Vegas, NV, 2000.
- [70] J. Kennedy and R. C. Eberhart, "A Discrete Binary Version of the Particle Swarm Algorithm," in *IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, FL, 1997.
- [71] E. C. Laskari, K. E. Parsopoulos and M. N. Vrahatis, "Particle Swarm Optimization for Integer Programming," in *Proceedings of the 2002 Congress on Evolutionary Computation*, Honolulu, HI, 2002.
- [72] X. Hu and R. Eberhart, "Multiobjective Optimization Using Dynamic Neighborhood Particle Swarm Optimization," in *Proceedings of the 2002 Congress on Evolutionary Computation*, Honolulu, HI, 2002.
- [73] C. A. Coello Coello, G. T. Pulido and M. S. Lechuga, "Handling Multiple Objectives With Particle Swarm Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 256 - 279, 2004.
- [74] T. Beielstein, K. E. Parsopoulos and M. N. Vrahatis, "Tuning PSO Parameters Through Sensitivity Analysis," *Interner Bericht des Sonderforschungsbereichs (SFB) 531 Computational Intelligence No.CI-124/02*, Affiliation: Universität Dortmund, January 2002.
- [75] R. V. O'Neill, R. H. Gardner and J. B. Mankin, "Analysis of Parameter Error in a Nonlinear Model," *Ecological Modelling*, vol. 8, pp. 297-311, 1980.

- [76] D. M. Hamby, "A Review of Techniques for Parameter Sensitivity Analysis of Environmental Models," *Environmental Monitoring and Assessment*, vol. 32, no. 2, pp. 135-154, 1994.
- [77] J. Golinski, "An Adaptive Optimization System Applied to Machine Synthesis," *Mechanism and Machine Theory*, vol. 8, no. 4, pp. 419-436, 1973 .
- [78] Y. Shi and R. C. Eberhart, "Empirical Study of Particle Swarm Optimization," in *Proceedings of the 1999 Congress on Evolutionary Computation*, Washington, DC, 1999.
- [79] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon and A. Abraham, "Inertia Weight Strategies in Particle Swarm," in *Third World Congress on Nature and Biologically Inspired Computing*, Salamanca, 2011.
- [80] J. Kennedy, "The Behavior of Particles," in *International Conference on Evolutionary Programming*, San Diego, CA, 1998.
- [81] R. C. Eberhart, P. Simpson and R. Dobbins, *Computational Intelligence PC Tools*, San Diego, CA: Academic Press Professional, 1996.
- [82] J. Barrera, O. Álvarez-Bajo, J. J. Flores and C. A. Coello Coello, "Limiting the Velocity in the Particle Swarm Optimization Algorithm," *Computación y Sistemas*, vol. 20, no. 4, p. 635–645, 2016.
- [83] G. Ciuprina, D. Ioan and I. Munteanu, "Use of Intelligent-Particle Swarm Optimization in Electromagnetics," *IEEE Transactions on Magnetics*, vol. 38, no. 2, pp. 1037 - 1040, 2002 .

- [84] J. J. Liang, A. K. Qin, S. Baskar and P. N. Suganthan, "Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multimodal Functions," *IEEE Transactions on Evolutionary Computation* , vol. 10, no. 3, pp. 281 - 295, 2006 .
- [85] A. Nickabadi, M. M. Ebadzadeh and R. Safabakhsh, "A Novel Particle Swarm Optimization Algorithm with Adaptive Inertia Weight," *Applied Soft Computing*, vol. 11, no. 4, p. 3658–3670, 2011.
- [86] X. Yang, J. Yuan, J. Yuan and H. Mao, "A Modified Particle Swarm Optimizer with Dynamic Adaptation," *Applied Mathematics and Computation*, vol. 189, no. 2, p. 1205–1213, 2007.
- [87] M. S. Arumugam and M. V. Rao, "On the Improved Performances of the Particle Swarm Optimization Algorithms with Adaptive Parameters, Cross-Over Operators and Root Mean Square (RMS) Variants for Computing Optimal Control of a Class of Hybrid Systems," *Applied Soft Computing*, vol. 8, no. 1, p. 324–336, 2008.
- [88] 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)*, vol. 39, no. 6, pp. 1362 - 1381, 2009.
- [89] S. Vakili, "Analysis of Water Colling Process of Steel Strups On Runout Table," in *Ph.D. Thesis*, University of British Columbia, 2011.
- [90] M. R. Alrasheed, C. W. de Silva and M. S. Gadala, "A Modified Particle Swarm Optimization Scheme and Its Application in Electronic Heat Sink Design," in *ASME/Pacific Rim Technical Conference and Exhibition on Packaging and Integration of Electronic and Photonic Systems, MEMS, and NEMS*, Vancouver, British Columbia, 2007.

- [91] K. H. Lee, S. W. Baek and K. W. Kim, "Inverse Radiation Analysis Using Repulsive Particle Swarm Optimization Algorithm," *International Journal of Heat and Mass Transfer*, vol. 51, no. 11-12, pp. 2772-2783, 2008.
- [92] Y. Wang, Z. Cai, Y. Zhou and Z. Fan, "Constrained Optimization Based on Hybrid Evolutionary Algorithm and Adaptive Constraint-Handling Technique," *Structural and Multidisciplinary Optimization*, vol. 37, no. 4, p. 395–413, 2009.
- [93] L. Wang and L.-p. Li, "An Effective Differential Evolution with Level Comparison for Constrained Engineering Design," *Structural and Multidisciplinary Optimization*, vol. 41, no. 6, p. 947–963, 2010.
- [94] E. Mezura-Montes, C. A. Coello Coello and J. Velázquez-Reyes, "Increasing Successful Offspring and Diversity in Differential Evolution for Engineering Design," *Proceedings of the Seventh International Conference on Adaptive Computing in Design and Manufacture (ACDM '06)*, p. 131–139, 2006.
- [95] H. Liu, Z. Cai and Y. Wang, "Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization," *Applied Soft Computing*, vol. 10, no. 2, pp. 629-640, 2010.
- [96] C. Zhang, Q. Lin, L. Gao and X. Li, "Backtracking Search Algorithm with Three Constraint Handling Methods for Constrained Optimization Problems," *Expert Systems with Applications*, vol. 42, no. 21, pp. 7831-7845, 2015.
- [97] H. Wang, Z. Hu, Y. Sun, Q. Su and X. Xia, "Modified Backtracking Search Optimization Algorithm Inspired by Simulated Annealing for Constrained Engineering Optimization Problems," *Computational Intelligence and Neuroscience*, vol. 2018, pp. 1-27, 2018.

- [98] F. Z. Huang, L. Wang and Q. He, "An effective co-evolutionary differential evolution for constrained optimization," *Applied Mathematics and Computation*, vol. 186, no. 1, p. 340–356, 2007.
- [99] C. A. Coello Coello and E. Mezura-Montes, "Constraint-Handling in Genetic Algorithms Through the Use of Dominance-Based Tournament Selection," *Advanced Engineering Informatics*, vol. 16, no. 3, pp. 193-203, 2002.
- [100] A. Sadollah, A. Bahreininejad, H. Eskandar and M. Hamdi, "Mine Blast Algorithm: A New Population Based Algorithm for Solving Constrained Engineering Optimization Problems," *Applied Soft Computing*, vol. 13, no. 5, pp. 2592-2612, 2013.
- [101] Q. He and L. Wang, "A Hybrid Particle Swarm Optimization with a Feasibility-Based Rule for Constrained Optimization," *Applied Mathematics and Computation*, vol. 186, no. 2, p. 1407–1422, 2007.
- [102] E. Sandgren, "Nonlinear Integer and Discrete Programming in Mechanical Design Optimization," *Journal of Mechanical Design*, vol. 112, no. 2, pp. 223-229, 1990.
- [103] P. E. Onate Yumbra, J. M. Ramirez and C. A. Coello Coello, "Optimal Power Flow Subject to Security Constraints Solved With a Particle Swarm Optimizer," *IEEE Transactions on Power Systems*, vol. 23, no. 1, pp. 33 - 40, 2008 .
- [104] J. S. Arora, Introduction to Optimum Design, 1st ed., New York: McGraw-Hill, 1989.
- [105] S. S. Rao, Engineering Optimization, 3rd ed., Wiley, 1996.
- [106] N. D. Lagaros, M. Papadrakakis and G. Kokossalakis, "Structural Optimization Using Evolutionary Algorithms," *Computers and Structures*, vol. 80, no. 7-8, p. 571–589, 2002.

- [107] S. L.A., "Structural Design by Systematic Synthesis," in *2nd Conference on Electronic Computation*, Pittsburgh, PA, 1960.
- [108] F. L. Stasa, *Applied Finite Element Analysis for Engineers*, New York, N.Y.: Holt, Rinehart and Winston, 1985.
- [109] D. Halliday, R. Resnick and J. Walker, *Fundamentals of Physics Extended*, 10th ed., New York: Wiley, 2013.

Appendices

Appendix A - MATLAB Code of the Speed-Reducer Design Problem

```
function [f] = obj_penal_speed_reducer(x)
global v
z = 0.7854*x(1)*(x(2)^2)*(3.3333*x(3)^2+ 14.9334*x(3)-43.0934)-
1.508*x(1)*(x(6)^2 +
x(7)^2)+7.4777*(x(6)^3+x(7)^3)+0.7854*(x(4)*(x(6)^2)+x(5)*(x(7)^2));
c(1) = (27/(x(1)*(x(2)^2)*x(3)))-1;
c(2) = (397.5/(x(1)*(x(2)^2)*(x(3)^2)))-1;
c(3) = ((1.93*x(4)^3)/(x(2)*x(3)*(x(6)^4)))-1;
c(4) = ((1.93*x(5)^3)/(x(2)*x(3)*(x(7)^4)))-1;
c(5) = (1/(110*x(6)^3))*(((745*x(4))/(x(2)*x(3)))^2) + 16.9 * 10^6)^(0.5)-1;
c(6) = (1/(85*x(7)^3))*(((745*x(5))/(x(2)*x(3)))^2) + 157.5 * 10^6)^(0.5)-1;
c(7) = ((x(2)*x(3))/40)-1;
c(8) = (5*x(2)/x(1))-1;
c(9) = (x(1)/(12*x(2)))-1;
c(10) = ((1.5*x(6)+1.9)/x(4)) -1;
c(11) = ((1.1*x(7)+1.9)/x(5)) - 1;
g = max(0, c);
for i = 1:length(c);
    if c(i) <= 0
        v(i) = 0;
    else v(i) = 1;
    end
end
f = z + 10e05*(sum(v) + sum(g));
```

Appendix B - MATLAB Code of the Pressure-Vessel Design Problem

```
function [f] = obj_penal_Pressure_Vessel(x)
global v
z = 0.6224*(0.0625*x(1))*x(3)*x(4) + 1.7781*(0.0625*x(2))*x(3)^2 + ...
    3.1661*(.0625*x(1))^2*x(4) + 19.84*(.0625*x(1))^2*x(3);
c(1) = (-0.0625*x(1) + 0.0193*x(3));
c(2) = (-0.0625*x(2) + 0.00954*x(3));
c(3) = -pi*x(3)^2*x(4) - (4/3)*pi*x(3)^3 + 1296000;
c(4) = (x(4)-240);
c(5) = 1 - x(1);
c(6) = 1 - x(2);
c(7) = 10 - x(3);
c(8) = 10 - x(4);
g = max(0, c);
for i = 1:length(c);
    if c(i) <= 0
        v(i) = 0;
    else v(i) = 1;
    end
end
f = z + 10e05*(sum(v) + sum(g));
```

Appendix C - MATLAB Code of the Tension/Compression Spring Design Problem

```
function [f] = obj_penal_spring(x)
global v
z = (x(3)+2)*x(2)*x(1)^2;
c(1) = (-x(2)^3*x(3))/(71785*x(1)^4) + 1;
c(2) = ((4*x(2)^2-x(1)*x(2))/(12566*(x(2)*x(1)^3-x(1)^4))) +
(1/(5108*x(1)^2)) - 1;
c(3) = 1 - ((140.45*x(1))/(x(2)^2*x(3)));
c(4) = ((x(1)+x(2))/1.5) - 1;
g = max(0, c);
for i = 1:length(c);
    if c(i) <= 0
        v(i) = 0;
    else v(i) = 1;
    end
end
f = z + 10e05*(sum(v) + sum(g));
```

Appendix D - MATLAB Code of the Welded Beam Design Problem

```
function [f] = obj_penal_welded_beam(x)
global v
z = 1.10471*x(1)^2*x(2) + 0.04811*x(3)*x(4)*(14 + x(2));
P = 6000; L = 14; E = 30*10^6; G = 12*10^6;
t_max = 13600; s_max = 30000; d_max = 0.25;
M = P*(L+(x(2)/2));
R = sqrt(x(2)^2/4 + ((x(1)+x(3))/2)^2);
J = 2*(sqrt(2)*x(1)*x(2)*((x(2)^2)/12+((x(1)+x(3))/2)^2));
t1 = P/(sqrt(2)*x(1)*x(2));
t2 = M*R/J;
t = sqrt((t1)^2 + 2*t1*t2*(x(2)/(2*R)) + (t2)^2);
s = 6*P*L/(x(4)*x(3)^2);
d = 4*P*L^3/(E*x(4)*x(3)^3);
P_c = (4.013*E*sqrt(x(3)^2*x(4)^6/36)/L^2)*(1-(x(3)/(2*L))*sqrt(E/(4*G)));
c(1) = t - t_max;
c(2) = s - s_max;
c(3) = x(1) - x(4);
c(4) = 0.10471*x(1)^2 + 0.04811*x(3)*x(4)*(14 + x(2)) - 5;
c(5) = 0.125 - x(1);
c(6) = d - d_max;
c(7) = P - P_c;
g = max(0, c);
for i = 1:length(c);
    if c(i) <= 0
        v(i) = 0;
    else v(i) = 1;
    end
end
f = z + 10e05*(sum(v) + sum(g));
```

Appendix E - MATLAB Code of the Three-Bar Truss Design Problem

```

function [f] = obj_three_bar(x)
global v
p = 70e3;
theta = pi/6;
E = 200e9;
Sa = 250e6;
rho = 7800;
g = 9.81;
z = rho*g*(x(4)*sqrt(x(1)^2+0.5^2) + x(5)*sqrt(x(2)^2+0.5^2)+ ...
x(6)*sqrt(x(3)^2+0.5^2));
K = 0;
Mass = 0;
for i = 1:3
    l(i) = sqrt(0.5^2+x(i)^2);
    m(i) = x(i)/l(i);
    n(i) = 0.5/l(i);
    a(i) = (E*x(i+3)/l(i));
    k = a(i)*[m(i)^2      m(i)*n(i);
              m(i)*n(i)  n(i)^2];
    d(i) = 1000*sqrt(x(i+3)*4/pi);
    M = rho*[1/3*x(i+3)*l(i)      0;
             0                  1/3*x(i+3)*l(i)];
    Mass = Mass + M;
    K = K + k;
end
P_vector = [p*cos(theta); -p*sin(theta)];
u = inv(K)*P_vector;
D = eig(K,Mass);
Freq = sqrt(min(D(1)));
c(1) = abs((-E/l(1))*(u(1)*m(1)+u(2)*n(1))) - Sa;
c(2) = abs((-E/l(2))*(u(1)*m(2)+u(2)*n(2))) - Sa;
c(3) = abs((-E/l(3))*(u(1)*m(3)+u(2)*n(3))) - Sa;
c(4) = E*x(4)*(u(1)*m(1)+u(2)*n(1))/l(1)-E*(pi)*x(4)^2/(4*l(1)^2);
c(5) = E*x(5)*(u(1)*m(2)+u(2)*n(2))/l(2)-E*(pi)*x(5)^2/(4*l(2)^2);
c(6) = E*x(6)*(u(1)*m(3)+u(2)*n(3))/l(3)-E*(pi)*x(6)^2/(4*l(3)^2);
c(7) = x(1) - x(2) + 0.1;
c(8) = x(2) - x(3) + 0.1;
c(9) = -x(4) + 3.2e-6;
c(10) = -x(5) + 3.2e-6;
c(11) = -x(6) + 3.2e-6;
c(12) = -Freq + 100;
c(13) = sqrt(u(1)^2 + u(2)^2) - 0.01;
g = max(0, c);
for i = 1:length(c);
    if c(i) <= 0
        v(i) = 0;
    else v(i) = 1;
    end
end
f = z + 10e05*(sum(v) + sum(g));

```

Appendix F - MATLAB Code of the Unique Adaptive Particle Swarm Optimization (UAPSO)

```
clc;
clear;
close all;
tic
%% Problem Definition
global v
nVar = 7;
VarSize = [1 nVar];
VarMin = [2.6, 0.7, 17, 7.3, 7.3, 2.9, 5.0];
VarMax = [3.6, 0.8, 28, 8.3, 8.3, 3.9, 5.5];
%% PSO Parameters
MaxIt = 3000;
MaxRun = 100;
nPop = 5;
cmax = 4.0;
cmin = 0.0;
%% Initialization
for k=1:MaxRun
    empty_particle.Position=[];
    empty_particle.Cost=[];
    empty_particle.Velocity=[];
    empty_particle.Violation=[];
    empty_particle.Best.Position=[];
    empty_particle.Best.Cost=[];
    empty_particle.Best.Velocity=[];
    empty_particle.Best.Violation=[];
    particle= repmat(empty_particle,nPop,1);
    GlobalBest.Cost=inf;
    for i=1:nPop
        particle(i).Position = unifrnd(VarMin,VarMax,VarSize);
        particle(i).Cost = CostFunction(particle(i).Position);
        particle(i).Violation = sum(v);
        particle(i).Velocity = zeros(VarSize);
        % Update Personal Best
        particle(i).Best.Position=particle(i).Position;
        particle(i).Best.Cost=particle(i).Cost;
        particle(i).Best.Velocity=particle(i).Velocity;
        particle(i).Best.Violation=particle(i).Violation;
        % Update Global Best
        if particle(i).Best.Cost<GlobalBest.Cost
            GlobalBest=particle(i).Best;
        end
    end
    BestCost=zeros(MaxIt,1);
    %% PSO Main Loop
    Best = zeros(MaxIt, nPop);
    Iteration = Best;
    r = Best;
    for it=1:MaxIt
        for i = 1:nPop
            Best(it,i) = particle(i).Best.Cost;
            Iteration(it,i) = particle(i).Cost;
            if it == 1
```



```

        Feasible_Solution(:,i) = particle(i).Cost;
    end
    if it > 1
        if particle(i).Violation == 0
            Feasible_Solution(it,i) = particle(i).Cost;
        else
            A = find(Feasible_Solution(:,i),1,'last');
            Feasible_Solution(it,i) = Feasible_Solution(A,i);
        end
    end
    end
    Global_Best (it,1) = GlobalBest.Cost;
    ES(it,i) = (Best(it,i)-
Global_Best(it,1))/Feasible_Solution(it,i);
    if ES(it,i)>=0.0 && ES(it,i)<=0.5
        c1(it,i) = ((cmax - cmin)*(MaxIt-it)/MaxIt)+ cmin;
        c2(it,i) = cmax - ((cmax - cmin)*(MaxIt-it)/MaxIt);
    elseif ES(it,i)>=0.5 && ES(it,i)<=1.0
        c1(it,i) = cmax - ((cmax - cmin)*(MaxIt-it)/MaxIt);
        c2(it,i) = ((cmax - cmin)*(MaxIt-it)/MaxIt)+ cmin;
    end
    Vmax = (VarMax);
    Vmin = - Vmax;
    for m = 1:nVar;
        if particle(i).Velocity(m) == 0
            if randn <= 0.5
                particle(i).Velocity(m) = rand*Vmax(m);
            else
                particle(i).Velocity(m) = rand*Vmin(m);
            end
        end
    end
    end
    % Update Velocity
    particle(i).Velocity = ES(it,i)*particle(i).Velocity ...
+ c1(it,i)*rand(VarSize).*(particle(i).Best.Position-
particle(i).Position) ...
+ c2(it,i)*rand(VarSize).*(GlobalBest.Position-
particle(i).Position)...
- (1-ES(it,i)*rand(VarSize).*(GlobalBest.Position-
particle(i).Best.Position);
    % Apply Velocity Limits
    particle(i).Velocity = max(particle(i).Velocity,Vmin);
    particle(i).Velocity = min(particle(i).Velocity,Vmax);
    % Update Position
    particle(i).Position = particle(i).Position +
particle(i).Velocity;
    % Apply Position Limits
    particle(i).Position = max(particle(i).Position,VarMin);
    particle(i).Position = min(particle(i).Position,VarMax);
    % Evaluation
    particle(i).Cost = CostFunction(particle(i).Position);
    Violation(it,i) = sum(v);
    particle(i).Violation =sum(v);
    % Update Personal Best
    if particle(i).Cost<particle(i).Best.Cost
        if particle(i).Violation == 0;
            particle(i).Best.Position=particle(i).Position;
            particle(i).Best.Cost=particle(i).Cost;

```

```

        particle(i).Best.Velocity=particle(i).Velocity;
        particle(i).Best.Violation=particle(i).Violation;
        % Update Global Best
        if particle(i).Best.Cost < GlobalBest.Cost;
            if particle(i).Violation == 0;
                GlobalBest = particle(i).Best;
            end
        end
    end
end
end
end
BestCost(it)=GlobalBest.Cost;
BestHistory(it, k) = BestCost(it);
disp(['Iteration ' num2str(it) ': Best Cost = '
num2str(BestCost(it))]);
end
BestSol (k)= GlobalBest;
Afields = fieldnames(BestSol);
Acell = struct2cell(BestSol);
sz = size(Acell);
Acell = reshape(Acell, sz(1), []);
Acell = Acell';
Acell = sortrows(Acell, 2);
Acell = reshape(Acell', sz);
BestSol = cell2struct(Acell, Afields, 1);
end
toc

```