TECHNISCHE UNIVERSITÄT MÜNCHEN

Modified Particle Swarm Optimizers and their Application to Robust Design and Structural Optimization

Bin Yang

Technische Universität München
Fakultät Bauingenieur- und Vermessungswesen

Lehrstuhl für Statik
Univ.-Prof. Dr.-Ing. Kai-Uwe Bletzinger
Arcisstr. 21
80333 München

Tel.: (+49 89) 289 - 22422
Fax: (+49 89) 289 - 22421

http://www.st.bv.tum.de

Statik

Lehrstuhl für Statik
der Technischen Universität München

Modified Particle Swarm Optimizers and their Application to Robust Design and Structural Optimization

Bin Yang

Vollständiger Abdruck der von der Fakultät für Bauingenieur– und Vermessungswesen der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor–Ingenieurs

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. M. Mensinger

Prüfer der Dissertation:

1. Univ.-Prof. Dr.-Ing. K.-U. Bletzinger
2. Univ.-Prof. Dr.-Ing. G. Müller

Die Dissertation wurde am 07.04.2009 bei der Technischen Universität München eingereicht und durch die Fakultät für Bauingenieur– und Vermessungswesen am 12.05.2009 angenommen.

Abstract

Many scientific, engineering and economic problems involve optimization. In reaction to that, numerous optimization algorithms have been proposed. Particle Swarm Optimization (PSO) is a new paradigm of Swarm Intelligence which is inspired by concepts from 'Social Psychology' and 'Artificial Life'. Essentially, PSO proposes that the co-operation of individuals promotes the evolution of the swarm. In terms of optimization, the hope would be to enhance the swarm's ability to search on a global scale so as to determine the global optimum in a fitness landscape. It has been empirically shown to perform well with regard to many different kinds of optimization problems. PSO is particularly a preferable candidate to solve highly nonlinear, non-convex and even discontinuous problems. The main ambition of this thesis is to propose two enhanced versions of PSO (Modified Guaranteed Convergence PSO (MGCPSO) and Modified *Lbest* based PSO (LPSO)) and to extend them to different areas of application.

MGCPSO is an extension of the *Gbest* based version of PSO and exhibits balanced performance between accuracy and efficiency in empirical numerical tests. It is applied to robust design with metamodel as well as structural sizing optimization in this work. In order to improve the efficiency of computing with regard to robust design, a mixed Fortran-Matlab program is developed as well as its corresponding parallel pattern. It obtains satisfying results for both optimization problems.

On the other hand, LPSO constitutes an enhanced *Lbest* based version of PSO whereby two LPSOs with two and three neighbour links are tested by means of the empirical benchmark test. Both demonstrate excellent global searching ability. For this reason, this algorithm is extended to problems of truss topological design. This type of optimization problems can be characterised as large-scale and non-convex. LPSO is very well suited to this particular field of optimization. Indeed, it achieves solutions that challenge the best ones ever found. Finally, MGCPSO is successfully applied to a structural sizing optimization problem 5.30).

Zusammenfassung

Viele wissenschaftliche, technische und wirtschaftliche Probleme sind Optimierungsprobleme. Da sie sich unter anderem hinsichtlich ihrer grundsätzlichen mathematischen Struktur unterscheiden, gibt es zahlreiche verschiedene Optimierungsalgorithmen. Die Partikel-Schwarm-Optimierung (Particle Swarm Optimization, PSO) ist ein neuartiges Paradigma der Schwarmintelligenz, das seine Motivation aus Konzepten der Sozialpsychologie und des Künstlichen Lebens gewinnt. PSO nimmt an, dass die Zusammenarbeit der Einzelnen die Entwicklung des Schwarmes wesentlich vorantreibt. Im Hinblick auf Optimierung wird erwartet, dass sowohl die Fähigkeit des Schwarmes optimale Lösungen zu finden wie auch die Qualität des Ergebnisses als die beste aller Möglichkeiten in einer „Fitness-Landschaft "verbessert werden. Es wurde empirisch gezeigt, dass mit PSO viele unterschiedliche Optimierungsaufgaben erfolgreich bearbeitet werden können. Insbesondere eignet sich PSO für stark nichtlineare, nicht-konvexe und auch diskontinuierliche Problemstellungen. Das Hauptziel dieser Dissertation ist es, zwei erweiterte Varianten, die sogenannte „modifizierte PSO mit garantierter Konvergenz"(MGCPSO) und die „modifizierte Lbest basierte PSO"(LPSO), zu entwickeln und für verschiedene Anwendungsbereiche zu etablieren.

MGCPSO ist eine Erweiterung der globalen Version von PSO. Wie empirische numerische Tests zeigen, stellt sie einen ausgewogenen Kompromiss zwischen Genauigkeit und Effizienz dar. Dieser Algorithmus kann für Aufgaben des „Robust Design"wie für Querschnittsoptimierung eingesetzt werden. Zur Verbesserung der Effizienz wurde eine gemischte Fortran-Matlab (grid computing) sowie eine parallelisierte Version entwickelt. Die damit erzielten Ergebnisse sind sehr zufriedenstellend.

LPSO verfolgt die Idee, die Nachbarbeziehungen zwischen Individuen des Schwarmes und die daraus abgeleiteten lokalen Eigenschaften der Optimierungsaufgabe zu nutzen. Dabei stellten sich die Varianten mit Gruppen aus zwei bzw. drei Nachbarn als besonders erfolgreich heraus und wurden an empirischen Benchmarks getestet. Insbesondere wird LPSO für die Topologieoptimierung von Fachwerken eingesetzt, eine nicht-konvexe Aufgabe mit vielen Optimierungsvariablen. LPSO erweist auch hier als sehr gut geeignet. Tatsächlich konnten für bekannte Testprobleme Lösungen gefunden werden, die sich mit den besten jemals gefundenen messen lassen können.

Acknowledgment

This thesis has been accomplished as part of my doctorate at the Chair of Structural Analysis, Department of Civil Engineering and Geodesy, the Technical University of Munich.

First and foremost, I owe Univ.-Prof. Dr.-Ing. Kai-Uwe Bletzinger, my supervisor, a debt of the most sincere gratitude for giving me the opportunity to undertake my doctorate at the Chair of Structural Analysis, as well as for his guidance and support - both of which I have found invaluable. What is more, I feel nothing but fortunate to have received the most excellent supervision from him in the past year and to have been able to build on his critical review of my work.

In addition, I would like to express my sincerest appreciation to Prof. Dr. Zhang Qilin from Tongji University (Shanghai, China) who encouraged me to pursue my doctoral studies at the Technical University of Munich in the first place. I wish to take this opportunity to thank him for all of his constructive suggestions regarding my research endeavours throughout the years.

Furthermore, it is with great pleasure that I express my thankfulness to both Univ.-Prof. Dr.-Ing. Gerhard Müller and Univ.-Prof. Martin Mensinger for the highly valuable instructions concerning the field of steel structure that they have given me, as well as for taking on the responsibility of reading and evaluating my thesis. I am also thoroughly grateful for their genuine interests in my work and for all of their insightful comments.

also wish to express my great appreciation for Prof. Kallmayer's and Herr Imhof's kind help with the preparatory work for my doctorate when I was still in China. Had it not been for their guidance, a smooth start of my studies in Germany would have been unthinkable.

Without the work of Bettina Hilliger and her boyfriend Nick, this thesis would be much poorer. I don't know of any other friend with enough patience to read this thesis and to do all the editing and correcting she has done.

Naturally, I wish to express my gratefulness to all of my colleagues for their constant encouragement, genuine help and all the time that they have spent helping me solve my problems throughout the challenging course of this thesis.

My warmest thank you also goes out to all of the friends that have made my stay in Munich a genuinely memorable and greatly enjoyable part of my life.

Last but certainly not least, I would also like to thank my parents and my girlfriend for the true patience and encouragement that they have shown me throughout the entirety of my doctoral studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In countless areas of human life, we attempt to exploit rigorously the limited amount of resources available to us so as to be able to maximize output or profit. In engineering design, for example, we are concerned with choosing design parameters that fulfill all the design requirements at the lowest cost possible. We deal in the same way with the task of allocating limited resources: Our main motivation is to comply with basic standards but also to achieve good economic results. Transforming problems of this nature into functions with corresponding constraints helps us to realize this aim. Optimization offers a technique for solving issues of this type because it provides a theoretical foundation, as well as methods for transforming the original problems into more abstract mathematical terms.

In mathematics, the term optimization refers to the study of problems in which one seeks to minimize or maximize a real function by systematically choosing the values of real or integer variables from within an allowed set. On one hand, a vast amount of research has been conducted in this area of knowledge with the hope of inventing an effective and efficient optimization algorithm. On the other hand, the application of existing algorithms to real projects has also been the focus of much research.

So far, the most commonly used optimization technique is called gradient algorithm which is based on gradient information. The latter, in turn, is derived from fitness functions and corresponding constraints. However, the acquisition of gradient information can be costly or even altogether impossible to obtain. Moreover, this kind of algorithm is only guaranteed to converge to a local minimal. But another kind of optimization algorithm - known as evolutionary computation (EC) - is not restricted in the aforementioned manner. Broadly speaking, EC constitutes a generic population-based metaheuristic optimization algorithm. Evolutionary algorithms tend to perform well with regard to most optimization problems. This is the case because they refrain from simplifying or making assumptions about the original form. Testament to this truth is its successful application to a great variety of fields, such as engineering, art, biology, economics, marketing, genetics, operations research, robotics, social sciences, physics, politics and chemistry. As a newly developed subset of EC, the Particle Swarm Optimization has demonstrated its many advantages and robust nature in recent decades. It is derived from social psychology and the simulation of the social behaviour of bird flocks in particular. Inspired by the swarm intelligence theory, Kennedy created a model which Eberhart then extended to formulate the practical optimization method known as particle swarm optimization (PSO) [72]. The algorithm behind PSO is based on the idea that individuals are able to evolve by exchanging information with their

neighbours through social interaction. This is known as cognitive ability. Three features impact on the evolution of the individual: Inertia (velocities cannot be changed abruptly), the individual itself (the individual could go back to the best solution found so far) and social influences (the individual could imitate the best solution found in its neighbour).

Generally, the PSO algorithm has the following advantages compared with other optimization algorithms:

- ◇ First of all, it is a simple algorithm with only a few parameters to be adjusted during the optimization process, rendering it compatible with any modern computer language.
- ◇ Second of all, it is also a very powerful algorithm because its application is virtually unlimited. Consequently, almost all kinds of optimization problems can be solved by PSO, normally in the original form.
- ◇ Last but not least, PSO is more efficient than other evolutionary algorithms due to its superior convergence speed.

These advantages result in its increasing popularity in the field of optimization since its proposal in 1995. Like other evolutionary algorithms, it can be applied to areas such as image and video analysis, signal processing, electromagnetic, reactive power and voltage control, end milling, ingredient mix optimization, antenna design, decision making, simulation and identification, robust design as well as structural optimization.

The main contributions of this thesis are the inventing of two modified particle swarm optimizers and their application to both robust design, as well as structural sizing and topological optimizations.

1.1 Statement of motivation

Clearly, there is an everlasting need to continually improve optimization algorithms with regard to all fields of optimization, simply because the complexity of problems that we attempt to solve is ever increasing. Since its introduction in 1995 [38, 72], particle swarm optimization has captured the attention of a great many researchers, mainly due to the algorithm's simplicity and efficiency in scanning large search spaces for optimal solutions. Inevitably, due to its belonging to the family of evolutionary computation (EC), PSO doesn't escape the problem of prematurity. Consequently, a lot of research has been conducted on how to avoid premature convergence. So far, concentrated effort has been made to develop efficient, robust and flexible algorithms, hoping to lead the particles to escape the local minima, especially in multi-modal problems. Because of the random features of PSO, most of its modifications were studied imperially with the aid of a series of testing functions without theoretical proof. The difficulty is that there doesn't exist a variant which could be a perfect extension of PSO. However, as much as this is a problem, it also challenges us to use our imagination in creative ways to improve the convergence performance of PSO.

PSO is easy to implement and for this reason has been successfully used to solve a wide range of optimization problems like pattern recognition, machine learning, task managing and so on. However, PSOs are not very widely used in two of the most popular fields of optimization: Robust design and topology. Besides, the applications are limited to simple cases. The "No Free Lunch Theorem" [136] points out that there is no universal optimization algorithm which can be used to solve all types of optimization problem. One way of addressing this problem would be to extend the application of PSO to the two aforementioned problems - of simple and complicated nature alike by choosing proper variants of PSO.

Although PSO has shown its potential with regard to solving a variety of problems, the execution time remains problematic in relation to matters of large-scale engineering, as well as other evolutionary algorithms. In order to address this issue, a high-performance PSO is necessary to alleviate the associated high cost of computing. Choosing appropriate parallel patters for the proposed PSOs constitutes yet another challenge for research. The ambition of this work is to develop proper parallel patterns in order to keep all processors working whilst the algorithm proceeds from one design iteration to the next, as efficiently as possible and without any loss in numerical accuracy. Finally, another task would be the improvement of the code's efficiency. Whilst it constitutes an interesting question and deserves mentioning for completion's sake, the matter is beyond the scope of this thesis.

1.2 Objectives

The primary objectives of this thesis can be summarized as follows:

- ◇ To study the existing modifications of PSO and to propose new variants which could potentially improve the performance of PSO, as well as the development of parallel modes.
- ◇ To study the proposed variants by means of numerical tests and to suggest suitable fields for their application.
- ◇ To apply them to proper examples and analyse their performance for further use.

1.3 Methodology

The greatest difficulty after developing an optimization algorithm is to verify its validity. In most cases of evolutionary algorithms, it is impossible to conjure up an analytical proof of the algorithm's convergence because too many indeterminable features exist. This is both enchanting and baffling to researchers. Even in the cases where this is possible, the proof usually demonstrates that the stochastic algorithm requires an infinite number of iterations to guarantee that it can determine the global optimum, which is impractical for normal applications. Therefore, the best way of validating an algorithm is of empirical nature. Empirical results can be obtained by using a well-known benchmark test that consists of a variety of functions with/without various constraints. In this way, a great amount of the challenges

that attach to an optimization algorithm are being dealt with, including high dimensionality, multi-modality and deceptive functions, allowing for a thorough testing of the algorithm. In order to test the stability of an algorithm, statistical analysis should also be used.

First of all, a great many of references concerning PSO will be investigated so as to ensure familiarity with the latest modifications of PSO. In addition, several representative variants and their respective extensions will also be selected and discussed. Following that, a summary will be made together with some suggestions on the improvement of PSO's performance. In light of these suggestions, two new and modified PSOs are proposed and the basic ideas behind them will be explained. Experimental results are obtained using a widely used benchmark test in order to show the accuracy and efficiency of the proposed algorithms, which are compared with Standard PSO. A large amount of evaluations of the objective functions relating to the PSO based optimizers is necessitated by the fact that we are concerned with an evolutionary algorithm. In order to utilize cluster to deduce computing time by parallel computing, relevant parallel patterns are also studied and tested with the same benchmark test. Furthermore, coefficients known as acceleration ratios are supplied in order to measure parallel efficiency.

Following preparatory work, these two variants are applied to different suitable fields that are robust design with meta models and structural sizing and topological optimization, respectively. With regard to **robust design**, two differing mathematical test functions are studied in order to investigate the performance of the proposed variant. Normally, programs of robust design are written in Matlab-code and the PSO with Fortran or C, C++. In order to reuse the existing code, it is necessary to use mixed programming, especially for parallel computing. The method implemented to realize this target is discussed first. Its superiority is demonstrated by contrasting the results from these two mathematical test functions with a more common optimization technique that is often used in robust design.

With regard to truss topological optimization, a series of very well respected equivalences of truss topological optimization with minimal compliance is introduced first. The successful methods to solve these equivalences are described in brief. The proposed variant (LPSO) is next tested with a benchmark test whose objective is to minimize structural compliance with continuous and discontinuous variables under a given volume. This kind of problem can be understood as a basic test for measuring the performance of an optimization technique that relates to the resolving of truss topological optimization issues. This means, that if and only if relatively satisfying results are achieved in this test, other researchers could be swayed to accept the results stemming from other topological problems by applying this very algorithm. Since in this case, the proposed variant passes the test quite well, it is also applied to the practical scenario of optimizing a truss bridge-like structure. The objective is to achieve a minimal weight, subject to a variety of constraints, such as displacement, stress and so on. In order to extend the MGCPSO's field of application, a structural sizing optimization is solved by means of it. Similar to the research work on robust design, corresponding parallel pattern is also studied.

Because of the stochastic nature of the proposed algorithms, all the presented results are averages and standard deviations from several simulations. Due to the expensive nature of the simulations, results were generally collected from ten or twenty runs.

1.4 Contribution

The main contributions of this thesis are:

- ◇ Two modified particle swarm optimizers are developed following a comparison of the representative variants of PSO. In addition, corresponding parallel patterns are proposed.
- ◇ These variants are tested by means of a well-known benchmark test. Moreover, suggestions based on the results are put forward.
- ◇ Finally, the two variants are applied to differing examples. The achievement of good results allows for their use in different applications.

1.5 Outline

Chapter 2 contains the history of PSO, as well as a detailed detailed description of the original PSO, especially in relation to information topology, parameter selection and so on. In order to facilitate a better understanding of this algorithm, comparisons with other evolutionary algorithms, such as the genetic algorithm and the ant colony algorithm, are made. Furthermore, this chapter deals with the numerous existing variants of PSO. A conclusion follows which summarizes the information on these variants and presents new ideas in the particle swarm paradigm and suggests new approaches for the field to improve the performance of the PSO. Lastly, an investigation of the various areas of application of PSO is included, based on an analysis of published materials on the application of PSO.

In chapter 3 two modified PSOs (MGCPSO (Modified Guaranteed Convergence PSO) and LPSO (Modified *Lbest* based PSO)) based on the aforementioned suggestions are proposed. To begin with, the basic idea behind MGCPSO is outlined, as well as that behind LPSO. This discussion comprises detailed descriptions and work flows relating to these two variants. Following that, in order to deduce the computing time, parallel patterns for MGCPSO and LPSO are developed and depicted in flow charts. The next section concerns itself with an empirical analysis of the modified algorithms and their corresponding parallel patterns that are put forward in this thesis. The next section is an empirical analysis of the modified algorithms as well as their corresponding parallel patterns presented in this thesis. Naturally, the results of the analysis are compared with those of the original PSO so as to highlight any improvements in terms of performance that MGCPSO and LPSO promoted. This, in turn, is followed by suggestions regarding possible fields of application.

In chapter 4, robust design and the application of MGCPSO thereto are discussed. To begin with, robust design is described in brief, including the concepts of robust design and the meta modelling technique. In turn, common optimization algorithm used in robust design with meta modelling is dealt with. Also, the main reason for the adoption of MGCPSO will be included. Although the common way of implementing robust design is to use matlab, it is not the most efficient strategy since the M-code in Matlab is not designed for the kind of

high performance computing that fortran is. A mixed programming of M-code and Fortran is then described as well as its parallel pattern. Based on the proposed computing pattern an evolutionary mode is also included to solve robust design problems with respect to parallel pattern with mixed M and Fortran codes. The chapter ends with comparisons of benchmark tests and an assessment of the efficiency of parallel computing.

The fifth chapter first provides an overview of "truss topology optimization" and a selective overview of the various techniques available for resolving this specific issue. Next, LPSO is applied and studied by means of a benchmark test of truss topological optimization with continuous or discontinuous variables. Moreover, following a comparison with the best results revealed so far through the Branch-and-Bound Algorithm [4], a near practical example is optimized under LPSO with differing structural constraints. Also, in oder in extend the application field of MGCPSO, a structural sizing optimization is solved by means of it. Last but not least, a range of concluding remarks as well as speed-up ratios are advanced based on all the investigations that were carried out.

Chapter 6 is the last part of this thesis which summarizes the achievements of my research work. Some topics for future research are also discussed.

Chapter 2

Theoretical Background & Particle Swarm Optimization

This chapter first reviews some basic concepts relating to optimization. A brief discussion of evolutionary algorithms is then provided. Lastly, the origins of PSO are discussed, followed by a study of the published modifications to the original PSO and some suggestions on further improvements.

2.1 Optimization

Optimization plays an important role in Computer Science, Artificial Intelligence, Operational Research and other such related fields. Nocedal and Wright at the beginning of their book [96] how optimization plays a part in our everyday lives:

> People optimize. Airline companies schedule crews and aircraft to minimize cost. Investors seek to create portfolios that avoid excessive risks while achieving a high rate of return. Manufacturers aim for maximum efficiency in the design and operation of their production processes. Nature optimizes. Physical systems tend to a state of minimum energy. The molecules in an isolated chemical system react with each other until the total potential energy of their electrons is minimized. Rays of light follow paths that minimize their travel time.

In order to use optimization successfully, we must first determine an objective through which we can measure the performance of the system under study. That objective could be time, cost, weight, potential energy or any combination of quantities that can be expressed by a single variable. The objective relies on certain characteristics of the system, called *variable* or *unknowns*. The goal is to find a set of values of the variable that results in the best possible solution to an optimization problem within a reasonable time limit. Normally, the variables are limited or constrained in some way. To illustrate this, the values of section areas in a structural optimization case cannot be negative and should belong to an interval that is predetermined by design standards.

The most important step in optimization is known as modelling which refers to the act of defining an appropriate objective, as well as variables and the constraints for the problem

at hand. If the model is too simple, it will not provide beneficial insights into the practical problem. On the other hand, if it is too complex, it may become too difficult to solve. If the numerical models involve only linear functions, then the optimization problems can be solved efficiently by a technique known as linear-programming [32]. Most problems addressed in this thesis constitute non-linear examples of optimization in which the model cannot be explicitly expressed in a set of non-linear functions or a combination of linear and non-linear ones. This is particularly the case with regard to the structural optimization cases. This field of optimization is naturally challenging.

In mathematical terms, optimization is the minimization or maximization of a function (called objective function or fitness function) subject to constraints on its variables. For simplicity's sake, hereafter all the optimization problems are assumed to find minima, a maximal problem could be transformed to minimal form by conveniently multiplying the objective function by -1. So that the optimization could be written as

$$\begin{array}{lll} \min\limits_{\mathbf{x}\in\mathbb{R}^n} & f(\mathbf{x}) & \\ \text{subject to} & g_i(\mathbf{x}) \leq 0 \quad ; \quad i = 1, \cdots, n_h = \mathbb{E} \\ & h_j(\mathbf{x}) = 0 \quad ; \quad j = 1, \cdots, n_g = \mathbb{I} \end{array} \tag{2.1}$$

where $\mathbf{x}$ is the vector of variables, also known as unknowns or parameters; f is the objective function with variables $\mathbf{x}$ to be optimized; $\mathbf{h}$ and $\mathbf{g}$ are vectors of functions representing equality constraints and inequality constraints respectively, and $\mathbb{E}$ and $\mathbb{I}$ are their relevant sets of indices.

2.1.1 Optimization Algorithms

Following the creation of the optimization model, the next task is to choose a proper algorithm to solve it. The optimization algorithms come from different areas and are inspired by different techniques. But they all share some common characteristics. They are iterative, they all begin with an initial guess of the optimal values of the variables and generate a sequence of improved estimates until they converge to a solution. The strategy used to move from one potential solution to the next is what distinguishes one algorithm from another. For instance, some of them use gradient based information and other similar methods (e.g. the first and second derivatives of these functions) to lead the search toward more promising areas of the design domain, whereas others use problem specific heuristics (e.g. accumulated information gathered at previous iterations). Broadly speaking, optimization algorithms can be placed in two categories: Local and global optimization.

2.1.2 Local Optimization

A local optimizer $\mathbf{x}^l$ could be denoted as

$$f(\mathbf{x}^l) \leq f(\mathbf{x}) \quad \forall \quad \mathbf{x} \in \mathbb{B} \quad \text{and} \quad \mathbb{B} \subset \mathbb{S} \subseteq \mathbb{R}^n \tag{2.2}$$

where $\mathbb{B}$ is a sub region of which $\mathbf{x}^l$ defines the local minimal, and $\mathbb{S}$ is the search domain. It is noted that if $\mathbb{S} = \mathbb{R}^n$, it is an unconstrained optimization problem. In general, the

design domain $\mathbb{S}$ can be divided into several sub regions $\mathbb{B}_i$, which satisfy $\mathbb{B}_i \bigcap \mathbb{B}_j = \emptyset$ when $i \neq j$, so that $\mathbf{x}^{l_i} \neq \mathbf{x}^{l_j}$, which means minimizer of each region B_i is unique, but the following relationship may be satisfied: $f(\mathbf{x}^{l_i}) = f(\mathbf{x}^{l_j})$. Any of the minimizers of $\mathbf{x}^{l_i}$ could be considered as a minimizer of $\mathbb{S}$, thus its representative solution is expressed by $f(\mathbf{x}^{l_i})$ named as local minimal, although they are only a local minimizer.

The local optimization algorithms always start with an initial point $\mathbf{z}^0 \in \mathbb{S}$ and try to find the best solution in the vicinity of the starting point. A local optimization algorithm should guarantee that it is able to find the minimizer $\mathbf{x}^l$ of the set $\mathbb{B}$ if $\mathbf{z}^0 \in \mathbb{B}$. So far many local optimization algorithms have been proposed and could be used to determine the local minimal effectively. The deterministic local optimization algorithms include simple Newton-Raphson algorithms, through Steepest Descent and its many variants, such as the Scaled Conjugate Gradient algorithm and the quasi-Newton family of algorithms. Some of the better known algorithms include Fletcher-Reeves (FR), Polar-Ribiere (PR), Davidon-Fletcher-Powell(DEP) and Broyden-Fletcher-Goldfarb-Shanno(BFGS) [96]. There even exists an algorithm that was designed for solving least-square problems alone, called the Levenberg-Marquardt (LM) algorithm.

2.1.3 Global Optimization

A global optimizer $\mathbf{x}_g$ could be denoted as

$$f(\mathbf{x}^g) \leq f(\mathbf{x}) \quad \forall \quad \mathbf{x} \in \mathbb{S} \subseteq \mathbb{R}^n \tag{2.3}$$

where $\mathbb{S}$ is the design domain. Throughout this thesis, the term global minimum refers to the value $f(\mathbf{x}^g)$ and $\mathbf{x}^g$ is denoted as global minimizer. A global optimization algorithm, much like the local optimization algorithms described above, also starts with an initial guessing position $\mathbf{z}^0 \in \mathbb{S}$. The global optimizer can be somewhat difficult to find because we don't normally tend to have a good picture of the overall shape of the objective functions f. This is especially true for multi-dimensional cases. Consequently, we can never be sure that the function does not take a sharp dip in some region that has not been searched by the algorithm. Figure 2.1 illustrates the differences between global minimizer $\mathbf{x}^g$ and local minimizer $\mathbf{x}^l$. For a true global optimizer, it will find $\mathbf{x}^g$ no matter where its starting point $\mathbf{z}^0$ is. However, there does exist an exceptional case in global optimization for which the finding of the global minimizer is definitely possible through a local optimization algorithm: In the case of convex functions, every local minimizer is also a global minimizer. This is formally expressed in a theorem.

Theorem 2.1 *When f is convex, any local minimizer $\mathbf{x}^l$ is a global minimizer of f.*

Before proofing that, the concept of convexity is firstly clearly defined here which could be applied both to design domains and to objective functions.

- $\mathbb{S} \in \mathbb{R}^n$ is a convex domain if the straight line segment connecting any two points in $\mathbb{S}$ lies entirely inside $\mathbb{S}$. Formally, for any two points $\mathbf{x} \in \mathbb{S}$ and $\mathbf{y} \in \mathbb{S}$, we have $\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in \mathbb{S} \qquad \forall \qquad \alpha \in [0,1]$

- f is a convex function if its domain is a convex domain and for any two points $\mathbf{x}$ and $\mathbf{y}$ in its domain, the figure of f always lies below the straight line connecting $(\mathbf{x}, f(\mathbf{x}))$ and $(\mathbf{y}, f(\mathbf{y}))$ in the space $\mathbb{R}^{n+1}$. Formally, this definition could be written as:

$$f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1-\alpha)f(\mathbf{y}) \qquad \forall \quad \alpha \in [0,1]$$

Here is the proof of theorem 2.1:
PROOF. Suppose that $\mathbf{x}^l$ is a local but not a global minimizer, so we could find a point $\mathbf{y}$ with $f(\mathbf{y}) < f(\mathbf{x}^l)$. Consider a line segment that joints $\mathbf{x}^l$ to $\mathbf{y}$, that is,

$$\mathbf{x} = \lambda\mathbf{y} + (1-\lambda)\mathbf{x}^l \qquad \text{for some} \quad \lambda \in (0,1] \tag{2.4}$$

Due to the convexity property of f, we have

$$f(\mathbf{x}) \leq \lambda f(y) + (1-\lambda)f(\mathbf{x}^l) < f(\mathbf{x}^l) \tag{2.5}$$

Any neighborhood $\mathbb{B}$ of $\mathbf{x}^l$ contains a piece of the line segment 2.4, so there will always be points $\mathbf{x}^l \in \mathbb{B}$ at which 2.5 is satisfied. Thus, $\mathbf{x}^l$ is not a local minimizer. This is in conflict with the assumption at the beginning of the proof, so that this theorem is proven correct.

Theorem 2.1 implies that the local optimization algorithm could be used to solve specific global optimization problems. Furthermore, if a non-convex problem could be transformed into convex form, it may be solved efficiently by a local optimization algorithm which, however, requires a strong background in mathematics.

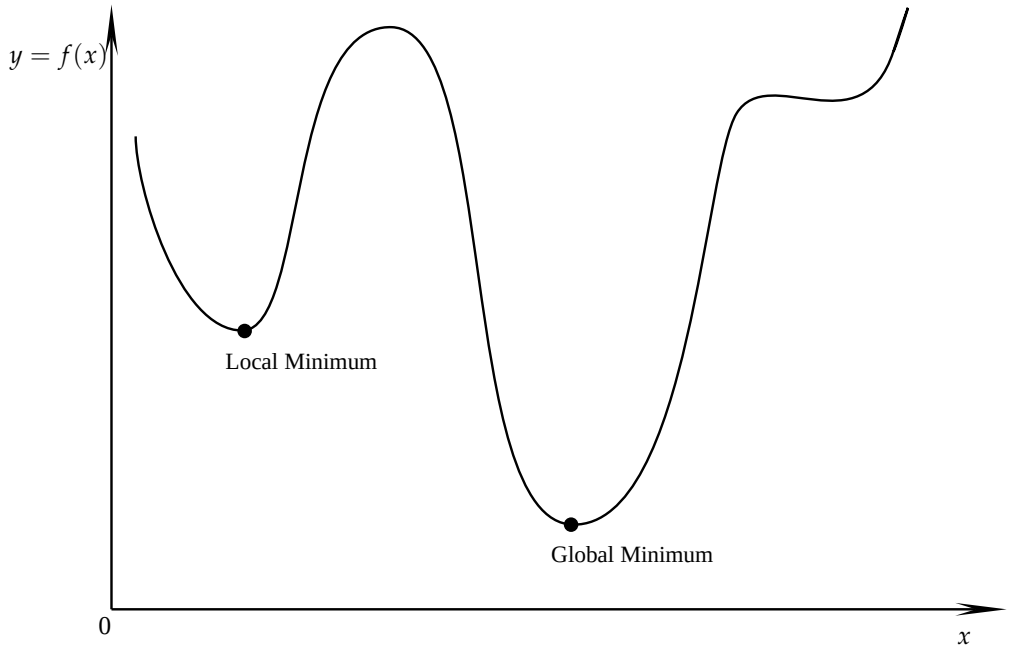


Figure 2.1: A multi-modal example for optimization

2.1.4 No Free Lunch Theorem

One of the most interesting theories in the field of optimization is the *no free lunch*(NFL) theorem proposed by Wolpert and Macready [136, 135]. This theorem not only states that for certain kinds of problems there exists a variety of different algorithms that are more or less applicable, it also argues that averaged over all possible problems or cost functions, the performance of all search algorithms is exactly the same.

The implications of this theorem are far reaching since it means that no algorithm can be designed that will ever be superior to a linear enumeration of the search space or even just better than a purely random search. In essence, no algorithm is deemed better, on average, than blind guessing. The theorem is only defined for finite search spaces, however, and it is not yet clear whether it also extends to infinite ones. All computer applications of search algorithms effectively operate in finite search spaces, although the theorem is also directly applicable to all other existing algorithms.

Despite the fact that the NFL theorem states that no one algorithm is better than another, considering every task that could possibly be imagined, it is perfectly plausible to think that one algorithm will be better suited than another to solving some certain subsets of set that we call "problems". The set of all functions over a finite domain includes the set of all the permutations of this domain. Many of these functions do not have compact descriptions and so they appear to be largely random. Most real-world functions, however, have some kind of structure and usually have compact descriptions. These types of functions form a rather small subset of the set of all functions. Recent research efforts can be divided into two categories. One is concerned with pinpointing the exact limitations of the various search strategies there are (i.e. trying to characterize the set of functions over which the NFL does not hold) and led to the development of improved versions of the sharpened versions of NFL [117], showing that it applies to a much smaller subset than initially thought. The other deals with finding the strengths of the various search strategies (i.e. trying to find the proper application field for these strategies in which the performance of these strategies is superior to others). To illustrate this, Christensen et al. [27] proposed a definition of a searchable function, as well as a general algorithm that probably performs better than random search on this set of searchable functions.

This thesis will side with the latter approach, assuming that it is possible to design algorithms that perform, on average, better than others (e.g. random search) over a limited subset of the set of all functions. No further attempt will be made to characterize this subset. Instead, numerical results will be used to show how the real-world problems could benefit from the improved algorithm.

2.2 Evolutionary Computation

Evolutionary Computation (EC) is a term that encompasses problem-solving paradigms whose key elements are the simulation of some of the known mechanisms of evolution. Generally, it differs from traditional search and optimization paradigms in three main ways by

1. Utilizing a population of individuals (potential solutions) in the search domain. Most traditional optimization algorithms move from one point to another in the search domain using some deterministic rule. One of the drawbacks of this kind of approach is the likelihood of getting stuck at a local optimum. EC paradigms, on the other hand, use a collection of points, called population. By inducing the evolutionary-like operators, they can generate a new population with the same number of members in each generation so that the probability of getting stuck is reduced.

2. Using direct fitness information instead of function derivatives or other related knowledge. EC paradigms do not require information that is auxiliary to the problem, such as function derivatives, but only the value of the fitness function, as well as the constraints functions for constrained problems. Thus fitness is a direct metric of the performance of the individual population member on the function being optimized.

3. Using probabilistic, rather than deterministic rules. These stochastic operators are applied to operations which lead the search towards regions where individuals are likely to find better values of the fitness function. So, the reproduction is often carried out with a probability which is dependent on the individual's fitness value.

In addition, EC implementations can encode the parameters in binary form or other symbols, rather than working with the parameters in their original form. For example, in genetic algorithms, the parameters of the problem are normally encoded in binary strings with a fixed length, which means the length does not vary during the optimization process. The length is determined by considering its maximum range and the precision required.

So far, the evolutionary computation field has been considered by many researchers to include four areas: Evolutionary Programming (EP), Evolutionary Strategies (ES), Genetic Algorithms (GAs) and Genetic Programming (GP). All of them belong to the general evolutionary framework. Differences can only be found in relation to the exact form of the operators, as well as the relationship between the size of the parent and offspring populations. Also, GP can be viewed as a specialized version of GA. Still, a universal procedure can be described that can be implemented no matter the type of algorithm:

1. Initializing the population in the search domain by seeding the population with random values.

2. Evaluating the fitness for each individual of the population

3. Generating a new population by reproducing selected individuals through evolutionary operations, such as crossover, mutation and so on.

4. Looping to step 2 until stopping criteria are satisfied.

The following subsections will provide brief introductions to the these subsets of evolutionary computation.

2.2.1 Evolutionary Programming

Evolutionary Programming (EP) is one of the four major Evolutionary computation (EC) paradigms. This term first appeared in the dissertation of Dr. Fogel entitled "*On the Organization of Intellect.*" in 1964 [51] which became the basis of the first book in the field of evolutionary programming that is known as "Artificial Intelligence through Simulated Evolution" in 1966 [52], co-authored by Owens and Walsh. This book is considered the landmark publication for Evolutionary Programming (EP) applications. In this book, finite state machines are used as predictors to generate symbol strings from Markov processes and non-stationary time series.

Evolutionary Programming is derived from the simulation of adaptive behaviour in evolution. It emphasizes the phenotype space of observable behaviour, meaning that it focuses on evolving behaviour and is thus suited for solving the problem at hand. In other words, it mimics "phenotypic evolution". This paradigm uses only selection and mutation without recombination. In essence, the generating of the new population for the next iteration relies exclusively on mutation for maintaining diversity in the whole population. Evolution strategy systems commonly generate far more offspring compared to parents (a ration of seven to one is common). In contrast, EP generates the same number of children as there are parents. Parents are selected to reproduce using a tournament method; their characters are mutated to generate children which are added to the current population. There are usually two methods to execute mutation on parents. One way is to assign a Gaussian random variable with zero mean and variance equal to the Euclidean distance between the parent and the origin on each parent vector component. The fitness of each of the children is then evaluated in the same way as the parents. Another way to perform mutation is to engage the process of self-adaptation, i.e. the standard deviations and rotation angles (if used) may also be adapted during the search based on their current values. As a result, the search adapts to the error surface contours.

Since the population doubles after mutation, to keep it constant, the process of selection is done in two specific ways. Traditionally, individuals are ranked according to the value of fitness function and the top half of the population is then preserved to the next iteration. Another method commonly used pits each individual against a number of others in a tournament. Each individual is assigned a number of marks according to the number of competitions won. An individual wins a competition with a probability equal to its fitness divided by the sum of fitness.

2.2.2 Evolution Strategies

Evolution strategies were first devised by Rechenber and Schwefel [105] as they were experimenting with mutation in order to find optimal physical configurations for hinged plates in wind tunnels. They noted that the standard gradient-descent algorithms could not be used to the specific sets of equations required for reducing wind resistance. They began to perturb their best solutions by mutation to search randomly in the nearby regions in the search domain and obtained good results. In early 1970s, Rechenberg published a book *"Evolution*

strategy: optimization of technical systems according to the principles of biological evolution" [106]. It is considered the foundation of this approach.

ES makes use of both mutation and recombination, searching the search domain and the strategy parameter domain simultaneously. The goal is to move the mass of the population towards the region of the landscape which so far has promoted the population the most. Whereas EP relies on a fixed mutation structure (i.e. each parent definitely generates one child in each iteration), ES only picks the 'best' individuals - "survival of the fittest" - (normally one-fifth best individuals of the population) to mutate. Mutation is performed by adding normally distributed random numbers to the parents' phenotypes, so that the next generation of children explores the regions which have proven promising for their parents.

The extent of the mutation is controlled in an interesting way in ES. Each individual is characterized by a set of features and a corresponding set of strategy parameters which are usually variances or standard deviations describing the variability of the normal distribution used to perturb the parents' features. For instance, random numbers can be generated based on the probability distribution with a mean of zero and a standard deviation defined by the strategy parameters and added to the parents' vector components which is recognized as a mutation operator. Of course the strategy parameters can also undergo the mutation process which is analogous to the self-adaptation process in evolutionary programming. In cases such as this, log normal distributions are sometimes used to define mutation standard deviations.

Two kinds of recombination operations are adopted by ES. The first and more common method is called discrete recombination in which the value of each component of the new individual is equal to the corresponding component of either parent. In intermediate recombination, each parameter value is set at a point between the values for the two parents, typically this point is set midway between those values. Research has shown that the best results are obtained by using discrete recombination for the parameter values and intermediate recombination for the strategy parameters. Bäck and Schwefel report that using strategy parameter recombination is mandatory for the success of any ES paradigm [12].

After new individuals are generated and added to the existing population, the next step is selection which is similar to evolutionary programming. Two different schemes are commonly used for selection called *comma* strategy (μ, λ) and *plus* strategy $(\mu + \lambda)$ respectively. In both cases, it is assumed that the number of children generated by μ parents is $\lambda > \mu$. The common interval is $\lambda/\mu \in [1, 7]$. For the *comma* strategy, the μ individuals with best fitness values are selected out of the children in order to replace the μ parents, so that in this case the μ parents do not undergo the selection procedure. The implication of this strategy is that good solutions (i.e. μ parents) from previous generation may be lost because the information of the parents are not stored. For the *plus* strategy, the μ parents and the λ children are put into a pool first, then μ members with the best fitness values are selected for the next generation. This strategy preserves the best solution found so far so that the value of the best individual of the population is decreasing monotonously. In ES, the *comma* version is regarded as the one that yields better performance [12]. The explanation is that deterioration of individuals could somehow provide greater progress in the future.

2.2.3 Genetic Algorithms

It is one of the most popular types of Evolutionary algorithms. To be more precise, it constitutes a computing model for simulating natural and genetic selection that is attached to the biological evolution described in Darwin's Theory which was first issued by Holland in 1975 [64]. In this computing model a population of abstract representations (called chromosomes or the genotype of the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem could result in better solutions, which are traditionally represented in binary form as strings comprising of 0s and 1s with fixed-length, but other kinds of encoding are also possible which include real-values and order chromosomes. In a very simple case, 128 could be represented by 10000000, whilst 127 could be written as 01111111. Of course, some GA techniques allow the user to specify the dynamic range and resolution for each variable. The program then will assign the proper number of bits and the coding. For example, if a variable stays an interval $[2.5, 6.5]$ and the solution to required to be precise to three decimal figures, the solution of this variable could be represented by a string 12 bit long, where the string of zeros represents the value of 2.5. By doing in this way, real-value problem could be involved in GA by a simple way.

Being a member of the family of evolutionary computation, the first step of GA is population initialization which is usually done stochastically. De Jong's dissertation [33] provides guidelines that are still observed: Start with a relatively high crossover rate, a relatively low mutation rate, and a moderately sized population-though what constitutes a moderately sized population is unclear. A population of normal size tends to comprise 20 and 200 individuals, depending primarily on the string length. The size normally increases linearly with individual string length rather than exponentially. However, the 'optimal' size depends on the problem. Occasionally, solutions may be "seeded" in areas where optimal solutions are likely to be found.

The GA usually uses three simple operators called selection, recombination (usually called crossover) and mutation. Selection is the step of a genetic algorithm in which a certain number of individuals is chosen from the current population for later breeding (recombination or crossover), the choosing rate is normally proportional to individual's fitness value. There are several general selection techniques. Tournament selection and fitness proportionality selection (also known as roulette-wheel selection) consider all given individuals. Other methods only choose those individuals with a fitness value greater than a given arbitrary constant. Crossover and mutation taken together is called reproduction. They are analogous to biological crossover and mutation respectively.

The most important operator in GA is crossover which refers to the recombination of genetic information during sexual reproduction. the child shares in common with it "parents" many characteristics. Therefore, in GAs, the offspring has an equal chance of receiving any given gene from either one parents because the parents' chromosomes are combined randomly. To date, there are many crossover techniques for organisms which use different data structures to store themselves, such like One-point crossover, two-point crossover, Uniform Crossover as well as Half Uniform Crossover. The probabilities for crossovers vary according to the problem. Generally speaking, values between 60 and 80 percent are typical for one-point crossover as well as two-point crossover. Uniform crossovers work well with

slightly lower probabilities on the other hand. The probability could also be altered during evolution. So a higher value might initially be attributed to the crossover probability. Then it is decreased linearly until the end of the run, ending with a value of one half or two-thirds of the initial value. Additionally for real-value cases, the crossover operation could be expressed as:

$$\mathbf{y}^1 = \lambda \mathbf{x}^1 + (1 - \lambda)\mathbf{x}^2 \tag{2.6}$$

$$\mathbf{y}^2 = \lambda \mathbf{x}^2 + (1 - \lambda)\mathbf{x}^1 \tag{2.7}$$

where $\mathbf{y}^1$ and $\mathbf{y}^2$ are two descendants created by two given parents $\mathbf{x}^1$ and $\mathbf{x}^2$, and $\lambda = U[0, 1]$.

Mutation is the stochastic flipping of chromosome bits that occurs each generation which is used to maintain genetic diversity from one generation of a population of chromosomes to the next. Mutation occurs step by step until the entire population has been covered. Again, the coding used is decisive. In the case of binary chromosomes, it simply flips the bit while in real-valued chromosomes a noise parameter $N[0, \sigma]$ is added to the value at that position. σ could be chosen in such way that it decreases in time, e.g. $\sigma = \frac{1}{1+\sqrt{t}}$, where t is the number of current iteration. The probability of mutation is normally kept as a low constant value for the entire run of the GA, such like 0.001. Despite of these evolutionary operators, in some cases a strategy called "elites" is used, where the best individual is directly copied to the next generation without undergoing any of the genetic operators.

In one single interaction, new parents are selected for each child and the process continues until a proper size of individuals for the new population is reached. This process ultimately results in the population of the new generation differing from the current one. Generally the average fitness of the population should be improved by this procedure since only the best organisms from the last generation are selected for breeding.

2.2.4 Genetic Programming

Genetic Programming (GP) is the fouth major area of evolutionary computation. The earliest works on the subject were done by Freidberg, Dunham and North in 1958 [53] and 1959 [54], when they tried to use a computer program to optimize another fix-length program. The first results on the GP methodology were reported by Smith in 1980 [122]. The current formal form of GP was first developed by John Koza in the late 1980s. Now GP has become one of the fastest-growing and most fascinating areas of evolutionary computation.

Unlike the other three branches that use binary strings or real-value variables it involves computers programs directly. Generally, they are represented as tree, graph or linear structures. Also, binary strings or real-value variables have a fixed length, whilst programs used in genetic programming use flexible sizes, shapes and are of different complexities. The goal of GP is to discover the one computer program that can provide exactly the desired output based on a given set of inputs. In other words, we are trying to teach computers how to solve problems automatically by means of GP.

In order to implement GP, the following preparatory steps are necessary:

1. The terminal set
2. The function set
3. The fitness measure
4. The system control parameters
5. The termination conditions

The terminal set compromises the system state variables and constants associated with the problem being solved. The only limitation to the function set is the programming language that is used to run the programs developed by GP. Each function in the function set should be assigned a certain number of arguments known as *arity* (e.g. Terminals are functions with arity 0). In one sense, the purpose behind GP can also be understood as finding a minimal set of functions that could accomplish whichever task is designated to it. As a result, the terminal set, as well as the function set could be identified in a very straightforward way for any specific problem. For some problems, the function set may consist of mathematical functions (such as *cos*, *sin*, etc.), arithmetic functions (addition, subtraction, multiplication, and division), as well as conditional operators (if-then-else, and iterative). The terminal set may consist of the program's external inputs variables and numerical constants. The task of choosing a function set is a very important step and heavily depends on the problem at hand.

The third step in preparation deals with the question of what needs to be done in order to successfully evaluate the fitness of the problem. The fitness measure is normally chosen to be inversely proportional to the error produced by the program output. Another way of measuring works much like a score system in which a program achieves a score in each iteration. The control parameters are normally the population size, the maximum number of iterations, the reproduction probability, crossover probability and the maximum depth of the trees. However, the most important one is the population size. The best way to choose a proper population size is by analyzing the problems fitness landscape. Terminating conditions may include the maximum number of generations to be run or a problem-specific success predicate. The final program normally tends to be the best program in terms of fitness measures created. It is noted that the last two preparatory steps are purely administrative.

After the completion of the five steps, the population is initiated by randomly generated computer programs composed of functions and terminals relevant to the problem at hand. Following that, crossover and reproduction are performed. It shall be noted that these two operators could be handled in a parallel manner by assigning two different probabilities to them, whilst the sum should be one. For example, if the probability of reproduction is 10 percent, the probability of mutation should be 90 percent. Depending on the measured fitness, a proper method for generating new generations will be determined, either reproduction or crossover. The decision is based on the probabilities as well as fitness. If reproduction is selected, it is often carried out by using roulette-wheel selection and coping the selected individuals into the new population. In the case of crossover, at first two parents are selected at random. Next, a new offspring program for the new population is created by randomly

recombining the chosen parts from two selected programs (crossover) or through random mutation of a part from one of the selected programs that is chosen at random (mutation). After the termination condition is met, the single best program is produced as the outcome of the run by the given population. If the run is successful, the result may be a solution (or approximate solution) to the problem.

Brief Summary to evolutionary computation Evolutionary Programming (EP), Evolutionary Strategies (ES), Genetic Algorithm (GA) and Genetic Programming (GP) share a great many similarities in common. For instance, all use a variation of selection based on the idea of the survival of the fittest. Additionally, all of them have many variants and are widely applied to a great number of fields. It shall be noted also that the boundaries between the different models of evolutionary computation are becoming increasingly indistinct. The tendency is to use hybrids of these approaches to solve more complex problems.

2.2.5 Swarm Intelligence

The term Swarm Intelligence (SI) denotes the fifth component of evolutionary computation and it naturally differs from the other four. It constitutes artificial intelligence based on the collective behaviour of decentralized, self-organized systems. In contrast to others, it doesn't employ evolutionary operations such as crossover, mutation and so on. This term was first used in the field of cellular robotic systems by Beni and Wang [133]. In 1999, Bonabeau, Dorigo and Theraulaz [22] extended the definition of swarm intelligence to refer to its more general sense. In their book, SI is defined as:

> Using the expression "swarm intelligence" to describe only this work seems unnecessarily restrictive: that is why we extend its definition to include any attempt to design algorithms or distributed problem-solving devices inspired by the collective behaviour of insect colonies and other animal societies.

From the aforementioned description it becomes clear that SI can be defined as any attempt to design algorithms or distributed problem-solving devices that are steered by the outcomes from the social interaction between local neighbours. A typical SI comprises a population of simple agents (i.e. a collection of interactive individuals, such as bees.) interacting locally with their neighbours. The agents act according to very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local interaction between such agents results in the emergence of complex global behaviour. Natural examples of SI include ant colonies, bird flocks, herds of animals, bacterial growth, and schools of fish. Moreover, the living space of SI could be extended from encompassing 3-dimensional space to including high-dimensional space in order to solve more complicated problems.

Swarm Intelligence-based techniques are now used in a number of applications. For example, the U.S. military is investigating swarm techniques for controlling unmanned vehicles; the European Space Agency is thinking about an orbital swarm for self assembly

and interferometry; NASA is investigating the use of swarm technology for planetary mapping; artists are using it as a means of creating complex interactive systems or simulating crowds; Tim Burton's "Batman Returns" applied swarm technology for a realistic depiction of the movements of a group of penguins using the "Boids" system as well as "The Lord of the Rings". The reason why swarm technology is particularly attractive is that it is cheap, robust, and simple. Also, swarm intelligence-related concepts and references can be found throughout popular culture, often in the shape of collective intelligence form of collective intelligence or a group mind involving far more agents than used in current applications.

2.2.5.1 Ant Colony Algorithm

As the first successful optimization algorithm based on SI, the Ant Colony Algorithm (ACO) was first introduced by Dorigo in 1992 in his PhD thesis [36]. This algorithm is a technique for solving optimization problems that relies on probability. The simulation of an ant colony is used as a model for finding satisfactory paths through graphs. In nature, ants look for food randomly but having found food, on their return to the nest, they will lay down pheromone trails along the path which dissipate over time and distance. If another ants are attracted by the trails and find the food guided by the trails, the trails will be enhanced, because they leave the same thing when they go back. Over time, however, the pheromone trail starts to evaporate thus reducing its attractiveness. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have for evaporation. In the case of a short path, by contrast, pheromone density is kept at a high level because there is no time for evaporation before the laying down of a new layer of pheromones. Pheromone evaporation also has the advantage of avoiding convergence to a local optimal solution. If there is no evaporation at all, the path chosen by the first ant would tend to be excessively attractive to the following ones, thus constraining the extend to which the solution space would be explored, resulting in the obtaining of a local minimum.

The most important operations in ACO are arc selection and Pheromone Update, which constitute the foundations of of the behaviour of ant colonies. Arc selection describes that an ant will move from node i to node j with probability $p_{i,j}$, which is defined as

$$p_{i,j} = \frac{\tau_{i,j}^{\alpha} \eta_{i,j}^{\beta}}{\sum \tau_{i,j}^{\alpha} \eta_{i,j}^{\beta}} \tag{2.8}$$

where $\tau_{i,j}$ represents the amount of pheromone in arc (i,j) and $\eta_{i,j}$ the desirability of arc (i,j), α and β are two adjustable parameters that control the relative weight of trail intensity and desirability.
Pheromone can be updated in

$$\tau_{i,j} = \rho \tau_{i,j} + \Delta \tau_{i,j} \tag{2.9}$$

where $\tau_{i,j}$ is the amount of pheromone in given arc (i,j), ρ is the rate of pheromone evaporation and Δ is the amount of pheromone deposited, typically given by

$$\Delta \tau_{i,j}^{k} = \begin{cases} 1/L_k & \text{if ant } k \text{ travels on arc } (i,j) \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

where L_k is the cost of the kth ant's tour (typically length).

Ant colony optimization algorithms have been successfully applied to produce near-optimal solutions to the traveling salesman problem, they have superior performance over simulated annealing and genetic algorithm approaches when the graph may change dynamically and the ant colony algorithm can be run continuously and adapt to changes in real time. The common variants include Elitist Ant System, Max-Min Ant System (MMAS) and ank-Based Ant System (ASrank).

2.2.5.2 Stochastic Diffusion Search

This is another successful approach of SI, which is introduced by Bishop in 1989 [20] as a population-based, pattern-matching algorithm. In SDS agents perform cheap, partial evaluations of a hypothesis (a candidate solution to the search problem). They then share information about hypotheses (diffusion of information) through direct one-to-one communication. As a result of the diffusion mechanism, high-quality solutions can be identified from clusters of agents with the same hypothesis. A very simple game (called the Restaurant Game) is a way of illustrating this algorithm in a very intuitive manner [34].

The Restaurant Game A delegation participates in a long-term conference in an unfamiliar town. Every night they need to find a restaurant to dine. Unfortunately there are a large number of restaurants in this town, each of which provides various meals. A requirement is that the delegation has to face a problem, such as finding restaurant where a maximum number of delegates would enjoy dining. It would be impossible to do a parallel exhaustive search with regard to all the possible combinations of restaurant and meal because this would simply take too long. A better choice is to employ a Stochastic Diffusion Search.

In order to accomplish this, each delegate selects a restaurant at random and proposes that it is the best one. At night each delegate tests his hypothesis by dining there and randomly selecting one meal on offer. The next morning during breakfast all of the delegates exchange their experiences concerning the last dinner. In the case that one individual did not enjoy last night's meal, he asks a randomly selected colleague to share with him his impressions of his dinner. If he receives positive feed-back, the disappointed diner will choose the recommended restaurant for his next dinner. Otherwise he simply selects another restaurant at random from those listed in the 'Yellow Pages'. The consequence of this approach is that very rapidly, a significant number of delegates will congregate around the restaurant that is perceived as 'best' in town.

SDS has been applied to problems as diverse as text search [20], object recognition [21], feature tracking [57], mobile robot self-localisation [13] and site selection for wireless networks [66]. In addition to the above techniques, efforts have been made in the past few years to develop new models for the swarm intelligence system, such as one based on *honey bee colony* [83] and *bacteria foraging* [82], as well as *particle swarm optimization* which will be discussed in the next section.

2.3 Origins of PSO

The Particle Swarm Optimization (PSO) algorithm belongs to category of the ECs for solving global optimization problems. Its concept was initially proposed by Kennedy as a simulation of social behaviour and the PSO algorithm was first introduced as an optimization method in 1995 by Kennedy and Eberhart [38, 72]. It can be used to solve a wide range of different optimization problems, including most of the problems that can be solved using Evolutionary Algorithms. The PSO is a stochastic algorithm that does not need gradient information derived from the fitness function. This allows the PSO to be used on functions where the gradient is either unavailable or expensive to compute. Similar to other evolutionary algorithms, it uses the potentially successful particles to search the design space, i.e. every particle has the ability to find the global optimum during the optimization process, even though its current position is the worst one among all of the particles in the given iteration.

2.3.1 Social behaviour

In biology, psychology and sociology social behaviour is behaviour directed towards society, or is taking place between members of the same species. In sociology, "behaviour" itself means an animal-like activity devoid of social meaning or social context, in contrast to "social behaviour" which features both. In a sociological hierarchy, social behaviour is followed by social action, which is directed at other people and is designed to induce a response. Further along this ascending scale are social interaction and social relation. Thus, social behaviour is a process of communication. Scientists have created computer simulations of various interpretations of the movement of organisms in a bird flock or fish school. A very influential simulation (*boids*) of bird flocks was done by Reynolds [108]. This simulation models very well the social behaviour of animals. Reynolds assumed in his paper that the bird flocks were driven by three concerns which can be described as follows:

1. **Separation**: Steer to avoid crowding local flock mates
2. **Alignment:** Steer towards the average heading of local flock mates
3. **Cohesion**: Steer to move towards the average position of local flock mates

Of course, this list is non-exhaustive and thus does not reflect the complexity of the behaviour of a flock of birds. Still, based on just these three rules, it is possible to simulate realistically the behaviour of a flock of birds and that of other creatures (including schools of fish and herds of animals) in computer graphics. This simple non-centralized algorithm is used in many animated cinematic sequences of flocks and herds. Compared with the social behaviour of the animals, human social behaviour is more complex, although it is governed by similar rules. Besides the physical motion, humans can adjust their beliefs, moving in a belief space. Meaning that two individuals could be apart in terms of their physical environment but they could occupy the same point in the belief space without collision. The idea is perfectly described in Kennedy's book [74].

> We are working towards a model that describes people's thinking as a social phenomenon. Thinking differs from the choreographed behaviour of fish and birds in two major ways. First, thinking takes place in a space of many more than three dimensions, as we have seen in our discussion of graphs and matrices, high-dimensional analogues of language, and neural nets. Second, when two minds converge on the same point in cognitive space, we call it agreement, not collision.

This cognitive feature of human social behaviour is highly intriguing and constitutes the main focus of this algorithm. It is often believed, and numerous examples from the natural world substantiate this claim, that a social sharing of information amongst the individuals of a population may provide an evolutionary advantage. This is the core idea behind the development of PSO [39].

2.3.2 Particle and Swarm

In discussing PSO, the concepts of the swarm and the particle are integral to an understanding of this algorithm. This section discusses the reason why this algorithm was named PSO. The term swarm (school, swarm or flock) relates to fish, insects, birds and micro-organisms, such as bacteria, and describes a behaviour of an aggregation (school) of animals of similar size and body orientation, generally cruising in the same direction. The term particle constitutes the basic component of the swarm and is mass-less and volume-less with its own velocity and acceleration.

The PSO's precursor was a simulator that was used to visualize the movement of a birds' flock [62, 107]. In this simulation, a point on the screen was designed as food, called the "cornfield vector" [72]; the idea was to simulate the birds' behaviour in finding food through social learning, by observing the behaviour of nearby birds who appeared to be near the food source. Matatić gave the following rules to describe behaviour of flock (or schools):

1. **Safe-Wandering**: The ability of a group of agents to move about while avoiding collisions with obstacles and each other.
2. **Dispersion**: The ability of a group of agents to spread out in order to establish and maintain some minimum inter-agent distance.
3. **Aggregation**: The ability of a group of agents to gather in order to establish and maintain some maximum inter-agent distance.
4. **Homing**: The ability to find a particular place or region.

Based on these rules, flock behaviour consists of *Safe-Wandering*, *Dispersion*, *Aggregation*, *Homing*. Kennedy and Eberhart simplified this model to the extent that it would only encompass homing and aggregation without dispersion and Safe-Wandering. The concepts of safe-wandering and dispersion are designed to force each member to move without collision. This is also covered by the idea of cognitive behaviour based on the social behaviour of

humans. Because of this, the two aforementioned concepts are made redundant. As a result, the population's activity resembles an intelligent swarm as defined by Mollonas which was noted when he developed his models for applications in artificial life. Based on his definition, only if a population satisfies the following fundamental principles, it can be called a Swarm [91].

1. Proximity Principle: the population should be able to carry out simple space and time computations.
2. Quality Principle: the population should be able to respond to quality factors in the environment.
3. Diverse Response Principle: the population should not commit its activity along excessively narrow channels.
4. Stability Principle: the population should not change its mode of behaviour every time the environment changes.
5. Adaptability Principle: the population should be able to change its behaviour mode when it is worth the computational price.

Eberhart proposed arguments in favour of the idea that the population used by the PSO possesses these properties. To illustrate this, the population is responding to the quality factors *pbest* (the best position in the bird's memory) and *gbest* (the best position of the flock). The allocation of responses between *pbest* and *gbest* ensures a diversity of response. The population changes its state (mode of behaviour) only when *gbest* changes, thus adhering to the principle of stability. The population is adaptive because it does change when *gbest* changes. Each bird can be thought of as a mass-less and volume-less particle moving in a certain region, having its own acceleration and velocity. Because of its cognitive ability, all birds could eventually occupy the same position (predefined task) after several iterations (which is inspired by human social behaviour). Similar concepts are also applied in some other fields, especially in computer graphics, which also use particle systems to describe the models used for creating effects like smoke or fire. This algorithm was named particle swarm optimization by Eberhart and Kennedy [38].

2.3.3 Initial PSO

The PSO was first introduced as an optimizer in 1995 [38]. The PSO is derived from a simplified version of the flock simulation. It also has features that are based upon human social behaviour (their cognitive ability). The PSO is initialized with a population of random solutions and the size of the population is fixed at this stage and is denoted as s. Normally, a search space should first be defined, e.g. like a cube of the form $[x_{min}, x_{max}]^D$for a Ddimensional case. Each particle is distributed randomly in the search region according to a uniform distribution which it shares in common with other algorithms of stochastic optimization. The position of any given particle in the search space is a vector representing a design variable for the optimization problem, which is also called a potential solution. In

addition, each particle has a velocity. This constitutes a major difference to other stochastic algorithms (e.g. GA). Here, the velocity is a vector that functions much like an operator that guides the particle to move from its current position to another potential improved place. All the particles' velocities are updated in every iteration. In order to describe the PSO conveniently, some symbols some symbols need to be defined:

- $\mathbf{x}^i(t)$: The position of particle i on time step t, i.e. it represents a Cartesian coordinates describing particle i's position in solution space.
- $\mathbf{v}^i(t)$: The velocity of particle i on time step t, i.e. it represents particle i's moving direction and its norm $\| \mathbf{v}^i(t) \|$ is corresponding step size.
- $\mathbf{p}^i(t)$: The best personal position of the particle i discovered so far, defined with dependence on time step t (for minimal case) as:

$$\mathbf{p}^i(t+1) = \begin{cases} \mathbf{p}^i(t) & \text{if } f(\mathbf{x}^i(t)) \geqslant f(\mathbf{p}^i(t)) \\ \mathbf{x}^i(t) & \text{if } f(\mathbf{x}^i(t)) < f(\mathbf{p}^i(t)) \end{cases} \tag{2.11}$$

- $\mathbb{S}_i$: A collection of particle i and its neighbor with size s_i , defined as:

$$\mathbb{S}_i \subseteq \mathbb{S} \mid \mathbf{x}^i \in \mathbb{S}_i, 2 \leq s_i \leq s \tag{2.12}$$

- $\hat{\mathbf{b}}^i(t)$: The best position in collection $\mathbf{s}^i$ on time step t, defined as:

$$\hat{\mathbf{b}}^i(t) = \{\hat{\mathbf{x}}(t) \mid \min f(\hat{\mathbf{x}}(t)), \quad \hat{\mathbf{x}}(t) \in \mathbb{S}_i\} \tag{2.13}$$

- $\mathbf{b}^i$: The so far discovered best position of particle i compared with its neighbors within $\mathbb{S}_i$ which is defined by a neighborhood topology (typical cases are shown in figure 2.3), defined as:

$$\mathbf{b}^i(t+1) = \begin{cases} \mathbf{b}^i(t) & \text{if } f(\hat{\mathbf{b}}^i(t+1)) \geqslant f(\mathbf{b}^i(t)) \\ \hat{\mathbf{b}}^i(t+1) & \text{if } f(\hat{\mathbf{b}}^i(t+1)) < f(\mathbf{b}^i(t)) \end{cases} \tag{2.14}$$

Now the initial PSO could be now denoted as:

$$\mathbf{v}^i(t+1) = \mathbf{v}^i(t) + C_1 R_1(\mathbf{p}^i(t) - \mathbf{x}^i(t)) + C_2 R_2(\mathbf{b}^i(t) - \mathbf{x}^i(t)) \tag{2.15}$$

$$\mathbf{x}^i(t+1) = \mathbf{x}^i(t) + \mathbf{v}^i(t+1) \tag{2.16}$$

where $R1$ and $R2$ are two independent random numbers selected in each step according to a uniform distribution in a given interval $[0, 1]$ and $C1$ and $C2$ are two constants which are equal to 2 in this initial version. The random number was multiplied by 2 to give it a mean of 1, so that particles would "overshoot" the target about half the time. Formula (2.15) clearly shows that the particle's velocity can be updated in three situations: The first one is known as the momentum part, meaning that the velocity cannot change abruptly from the velocity of the last step; and that it could be scaled by a constant as in the modified versions of PSO. The second one is called "memory" part and describes the idea that the individual learns from its flying experience. The last one is known as the "cognitive" part which denotes the concept that particles learn from their group flying experience because of collaboration. An intuitive example is shown in figure 2.2. According to the description above, the whole workflow of the standard particle swarm optimization is shown below:

1. Initialize the swarm by assigning a random position in the design domain to each particle.
2. Evaluate the fitness function over the swarm
3. For each particle, update its $\mathbf{p}^i$ and $\mathbf{b}^i$
4. Update the velocities and positions of all particles
5. Repeat steps 2-4 until a stopping criterion is satisfied and output the result.

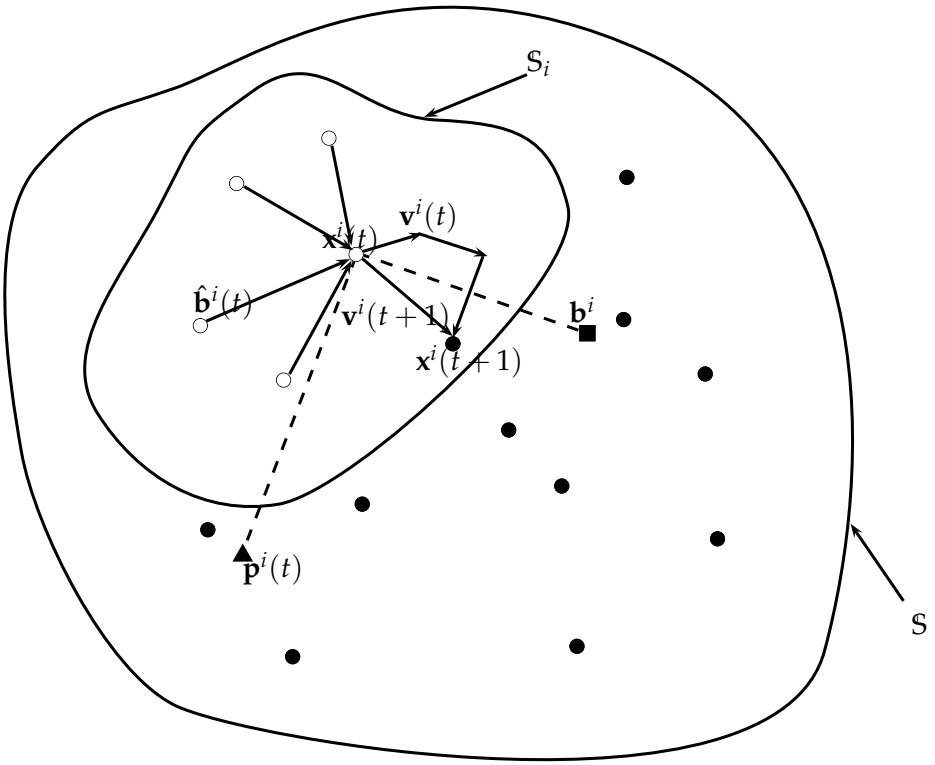


Figure 2.2: An demonstration of PSO for *LBEST* model

Here is the corresponding Pseudo code:
Create and **initialize** a Swarm with s particles and their corresponding collection including neighbors $\mathbb{S}_i$
Repeat:

for each particle $i \in \mathbb{S}$

update $\mathbf{v}^i(t+1)$ using formula (2.15)
update $\mathbf{x}^i(t+1)$ using formula (2.16)
if $f(\mathbf{x}^i(t+1)) < \mathbf{p}^i(t)$

then $\mathbf{p}^i(t+1) = \mathbf{x}^i(t+1)$

else if $f(\mathbf{x}^i(t+1)) \geq \mathbf{p}^i(t)$

then $\mathbf{p}^i(t+1) = \mathbf{p}^i(t)$

end if

Set $\hat{\mathbf{b}}^i(t+1) = \{\hat{\mathbf{x}}(t+1) \mid \min f(\hat{\mathbf{x}}(t+1)), \quad \hat{\mathbf{x}}(t+1) \in \mathbb{S}_i\}$

if $\hat{\mathbf{b}}^i(t+1) < \mathbf{b}^i(t)$

then $\mathbf{b}^i(t+1) = \hat{\mathbf{b}}^i(t+1)$

else if $\hat{\mathbf{b}}^i(t+1) \geq \mathbf{b}^i(t)$

then $\mathbf{b}^i(t+1) = \mathbf{b}^i(t)$

end if

Until stopping criterion is satisfied

Output result

2.3.4 Parameter selection

After the PSO was issued, several considerations has been taken into account to facilitate the convergence and prevent an "explosion" of the swarm. These considerations focus on limiting the maximum velocity, selecting acceleration constants and constriction factor. These features constitute the fundamental improvements of this algorithm and they are also adopted by the PSO's variations. Details are discussed in the following:

Selection of maximum velocity: At each step of the iteration, all particles move to a new place, the direction and distance is defined by their velocities. It is not desirable in practice that many particles leave the search space in an explosive manner. Formula (2.15) shows that the velocity of any given particle is a stochastic variable and that it is prone to create an uncontrolled trajectory, allowing the particle to follow wider cycles in the design space, as well as letting even more escape it. In order to limit the impact of this phenomena, particle's velocity should be clamped into a reasonable interval. Here the new constant v_{max} is defined:

If $v_j^i > v_{max}$, then $v_j^i = v_{max}$

If $v_j^i < -v_{max}$, then $v_j^i = -v_{max}$

Normally, the value of v_{max} is set empirically, according to the characteristics of the problem. A large value increases the convergence speed, as well as the probability of convergence to a local minimum. In contrast, a small value decreases the efficiency of the algorithm whilst increasing its ability to search. Empirical rules require that for any given dimension, the value of v_{max} could be set as half range of possible values for the search space. Research work from Fan and Shi [44] shows that an appropriate dynamically changing v_{max} can improve the performance of the PSO. Moreover, to ensure uniform velocity over all dimensions, Abido [1]

proposed a formula to define $\mathbf{v}_{max}$:

$$\mathbf{v}_{max} = (\mathbf{x}_{max} - \mathbf{x}_{min})/N \tag{2.17}$$

where N is a integer selected by the user and $\mathbf{x}_{max}$, $\mathbf{x}_{min}$ are maximum and minimum values found so far, respectively.

Adding Inertia Weight: Shi and Eberhart [120] proposed a new parameter ω for the PSO, named inertia weight, in order to better control the scope of the search, which multiplies the velocity at the previous time step, i.e., $\mathbf{v}^i(t)$. The use of the inertia weight ω improved performance in a number of applications. This parameter can be interpreted as an "inertia constant", formula (2.15) is now updated bellow:

$$\mathbf{v}^i(t+1) = \omega\mathbf{v}^i(t) + C_1R_1(\mathbf{p}^i(t) - \mathbf{x}^i(t)) + C_2R_2(\mathbf{b}^i(t) - \mathbf{x}^i(t)) \tag{2.18}$$

This inertia weight can either be a constant or a dynamically changed value. If $\omega = 1$, formula (2.18) turns back to the original form (2.15). Essentially, this parameter controls the exploration of the search space, so that a high value allows particles to move with large velocities in order to find the global optimum neighborhood in a fast way and a low value can narrow the particles' search region. Research has been taken into account in order to find a suitable set of ω. So far there are two common strategies to choose the value of ω:

1. Linear strategy: The value of ω is decreased from a higher initial value (typically 0.9) to a lower value (typically 0.4) linearly as the iteration number increases. It has shown good performance in some applications. As the value of ω is decreased, the search model of particles is also transformed from an exploratory mode to an exploitive mode. The main disadvantage of this strategy is that once the inertia weight is decreased , the swarm loses its ability to search new areas because it is impossible to recover its exploration mode.

2. Random strategy: the value of ω comprises two parts, a constant (typically 0.5) and a random value distributed in $[0, 0.5]$. The constant part ensures the particles' basic search ability within an exploitive mode, the random part ensures that the particle can shift its search mode between exploratory mode and exploitative mode randomly. Using this strategy, particles can search the design domain more flexibly and widely.

Selecting acceleration constants: Acceleration constants $C1$ and $C2$ in formula (2.15) control the movement of each particle. Small values limit the movement of the particles and large values may cause the particle to diverge. These values are normally selected empirically. Ozcan and Mohan proposed some suggestions on how to choose $C1$ and $C2$ after several experiments for the special case of a single particle in a one-dimensional problem space [99]. In such a case, the two acceleration constants are considered as a single acceleration constant $C = C1 + C2$. Experiments show that if the value of this acceleration constant increases, the frequency of the oscillations around the optimal point increases too. For smaller values of C, the particle's trajectory follows a wide path. The trajectory goes to infinity for values of C that are greater than 4.0.In general, the maximum value of C should be 4.0. $C1 = C2 = 2.0$ has been proposed as a good starting point. Notably, $C1$ and $C2$ should not always be

equal to each other since the "weights" for individual, as well as group experiences can vary depending on the characteristics of the problem.

Parameter selection strategy: Perex and Behdinan put forward a guide on how to select Inertia Weight and acceleration constants by producing a convergence and stability analysis of the standard PSO [101]. Let us recall formula(2.18) and(2.16). The following formula(2.19) is obtained by inducing formula(2.15) into formula(2.16) and rearranging the position term $\mathbf{x}^i(t)$

$$\mathbf{x}^i(t+1) = \mathbf{x}^i(t)(1 - C_1R_1 - C_2R_2) + \omega\mathbf{v}^i(t) + C_1R_1\mathbf{p}^i(t) + C_2R_2\mathbf{b}^i(t) \tag{2.19}$$

Similarly, by rearranging the position term $\mathbf{x}^i(t)$ of formula(2.15), formula(2.20) was obtained:

$$\mathbf{v}^i(t+1) = -\mathbf{x}^i(t)(C_1R_1 - C_2R_2) + \omega\mathbf{v}^i(t) + C_1R_1\mathbf{p}^i(t) + C_2R_2\mathbf{b}^i(t) \tag{2.20}$$

And finally, formula(2.19) and formula(2.20) are rearranged in matrix form:

$$\begin{bmatrix} \mathbf{x}^i(t+1) \\ \mathbf{v}^i(t+1) \end{bmatrix} = \begin{bmatrix} 1 - C1R1 - C2R2 & \omega \\ -C1R1 - C2R2 & \omega \end{bmatrix} \begin{bmatrix} \mathbf{x}^i(t) \\ \mathbf{v}^i(t) \end{bmatrix} + \begin{bmatrix} C1R1 & C2R2 \\ C1R1 & C2R2 \end{bmatrix} \begin{bmatrix} \mathbf{p}^i(t) \\ \mathbf{b}^i(t) \end{bmatrix} \tag{2.21}$$

which could be considered as discrete-dynamic system for the PSO algorithm where $[\mathbf{x}^i(t), \mathbf{v}^i(t)]^T$ could be considered as the state subject to an external input $[\mathbf{p}^i(t), \mathbf{b}^i(t)]^T$. The first matrix could be denoted as the dynamic matrix and the second one denoted as the input matrix based on the above assumption. Here, a simplification is made in order to do research on the particle's convergence behaviour, it is assumed that:

1. $[\mathbf{p}^i(t), \mathbf{b}^i(t)]^T$ is constant;
2. there is no external excitation;
3. other particles do not find better positions.

If the number of the iterations tends to be infinite, convergence $[\mathbf{x}^i(t+1), \mathbf{v}^i(t+1)]^T = [\mathbf{x}^i(t), \mathbf{v}^i(t)]^T$ might be satisfied and formula (2.21) will be reduced as:

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -C1R1 - C2R2 & \omega \\ -C1R1 - C2R2 & \omega - 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}^i(t) \\ \mathbf{v}^i(t) \end{bmatrix} + \begin{bmatrix} C1R1 & C2R2 \\ C1R1 & C2R2 \end{bmatrix} \begin{bmatrix} \mathbf{p}^i(t) \\ \mathbf{b}^i(t) \end{bmatrix} \tag{2.22}$$

Which is satisfied only when $\mathbf{v}^i(t+1) = 0$ and both $\mathbf{x}^i(t+1)$ and $\mathbf{p}^i(t+1)$ are located on $\mathbf{b}^i(t+1)$. With regard to defining an equilibrium point, it is important to note that this position is not necessarily a local or global optimum. Such point will move towards the optimum if the external excitation could stimulate the dynamic system and better results can be found during the optimization process. The dynamic matrix characteristic of formula (2.21) could be arrived at in the following way:

$$\lambda^2 - (\omega - C1R1 - C2R2 + 1)\lambda + \omega = 0 \tag{2.23}$$

where the eigenvalues are given as:

$$\lambda_{1,2} = \frac{1 + \omega - C1R1 - C2R2 \pm \sqrt{(1 + \omega - C1R1 - C2R2)^2 - 4\omega}}{2} \tag{2.24}$$

The necessary and sufficient condition for a stable discrete-dynamic system is that all eigenvalues λ are derived from the dynamic matrix and that they stay inside a unit circle around the origin on the complex plane, in other words,$\mid \lambda_{i=1,\cdot,n} \mid < 1$. Based on this necessary and sufficient condition the following relationships can be arrived at by analysing formula (2.24)

$$\begin{array}{l} C1R1 + C2R2 > 0 \\ \frac{C1R1+C2R2}{2} - \omega < 1 \\ \omega < 1 \end{array} \tag{2.25}$$

A practical condition could be derived from:

$$\begin{array}{l} 0 < C1 + C2 < 4 \\ \frac{C1+C2}{2} - 1 < \omega < 1 \end{array} \tag{2.26}$$

It must be noted that formula (2.26) is only a necessary condition for formula (2.25). By applying formula (2.26), a set of ω, $C1$ and $C2$ is quickly achieved, which also satisfies formula (2.25), so that the PSO algorithm has guaranteed convergence to an equilibrium point. This serves as a guideline for selecting parameters for the PSO algorithm.

Constriction Factor: When the particle swarm algorithm is run without restraining the velocity in some way, the system simply explodes after a few iterations. Clerc and Kennedy [29] induced a constriction coefficient χ in order to control the convergence properties of a particle swarm system. A very simple model using a constriction factor is shown bellow:

$$\mathbf{v}^i(t+1) = \chi \left[\mathbf{v}^i(t) + C_1 R_1 (\mathbf{p}^i(t) - \mathbf{x}^i(t)) + C_2 R_2 (\mathbf{b}^i(t) - \mathbf{x}^i(t)) \right] \tag{2.27}$$

where

$$\chi = \frac{2}{\mid 2 - C - \sqrt{C^2 - 4C} \mid}, \quad C1 + C2 = C > 4.0 \tag{2.28}$$

The constriction factor results in convergence over time; the amplitude of the trajectory's oscillations decreases over time. Note that as C increases above 4.0, χ gets smaller. For instance, if $C = 5$ then $\chi \approx 0.38$ from formula (2.28), which causes a very pronounced damping effect. A normal choice is that C is set to 4.1 and the constant χ is thus 0.729, which works fine.

The advantage of using constriction factor χ is that there is no need to use v_{max} nor to guess the values for any parameters which govern the convergence and prevent explosion. The disadvantage is that the particles may follow wider cycles and may not converge when $\mathbf{p}^i$ is far from $\mathbf{b}^i$(i.e. these are two different regions), especially in multi-modal optimization problems.

2.3.5 *Gbest* and *Lbest* model

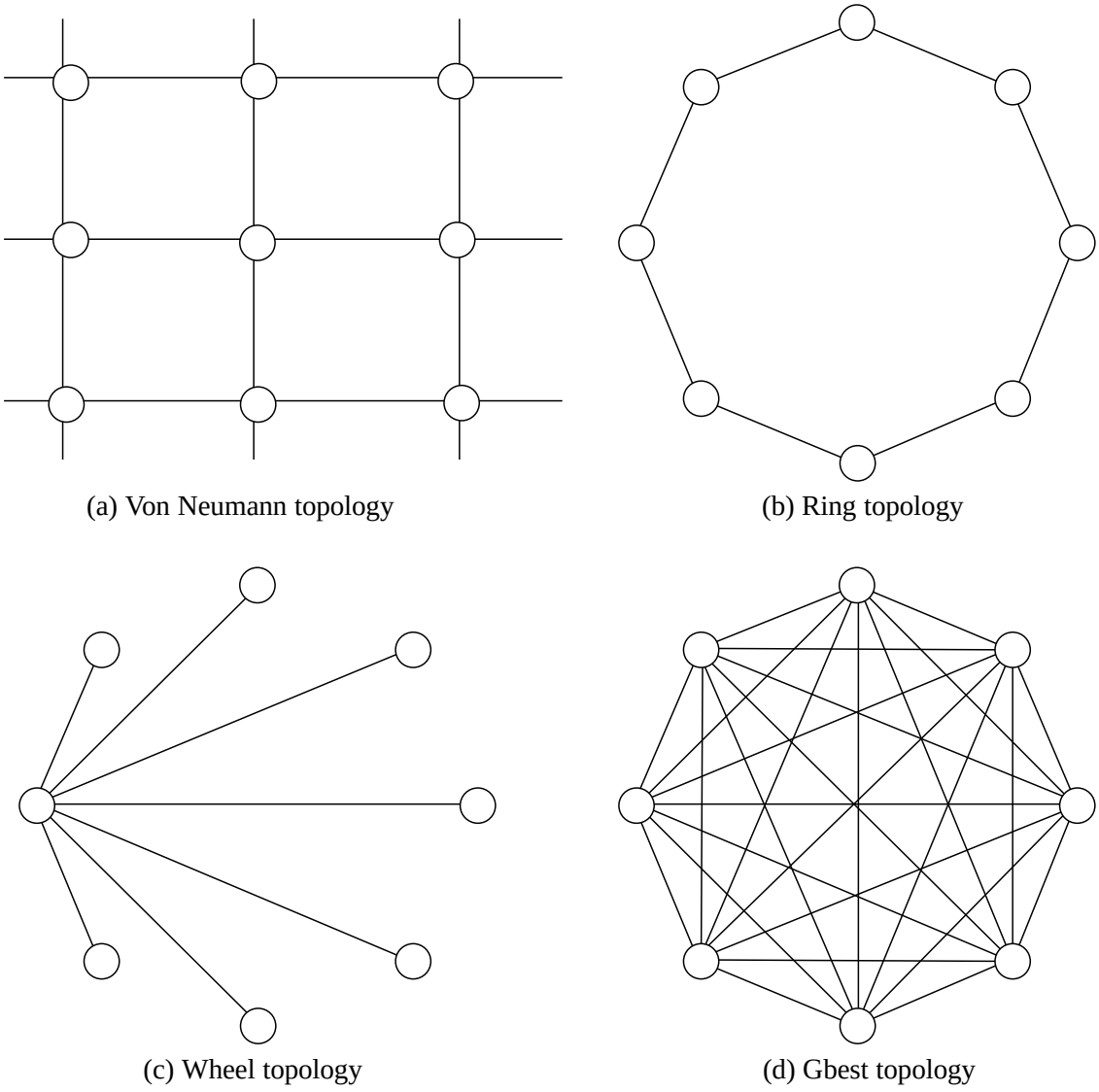
Historically, particles have been studied in two types of neighbourhood - the *Gbest* and the *Lbest*. In this thesis, particle i's neighborhood is referred to as a relationship between particle i and its neighbors and the corresponding sub-domain $\mathbb{S}^i$ which is constituted by them. For instance, inside $\mathbb{S}^i$, particle i can communicate with its neighbors whereas its neighbors cannot communicate with each other. The *Gbest* model connects all members of the population to one another, so that each individual is attracted to the best solution $\mathbf{b}$ found by a member of the swarm, i.e. all of the particles are pushed towards this position, if $\mathbf{b}$ can not be updated regularly, the swarm may converge prematurely. This structure amounts to the equivalent of a fully connected social network as shown in figure 2.3(d). It allows each individual to compare the performance of other members of the population. This means that all particles are interconnected in terms of information so that the best position can be communicated inside the entire swarm immediately after being found by a particle. In the *Lbest* model each individual is influenced by the best performances of its $\mathbf{s}_i$ neighbours. This is defined in the topology technique. There are three common topologies for the *Lbest* model: Von Neumann topology (seen figure 2.3(a)) in which particles are connected using a grid network (2-dimensional lattice) where each particle is connected to its four neighbor particles (above, below, right and left particles); Ring topology (seen figure 2.3(b)) in which each particle is connected with two neighbors; Wheel topology (seen figure 2.3(c)) in which the particles are isolated from one another and all the information is communicated to a focal individual. Note that once the neighborhood topology is created, it will not be changed during optimization procedure. The *Lbest* model tried to prevent premature convergence by maintaining diversity of potential problem solutions. Whilst it can search the design space sufficiently, its convergence speed is relatively slow compared to the *Gbest* model. Note that the *Gbest* model can be regarded a special case of the *Lbest* model with $\mathbb{S}_i = \mathbb{S}, \quad i \in \{1, \cdots, s\}$. This is illustrated in figure 2.2, a sub-domain $\mathbb{S}_i$ can be seen as a subset including particle i and its $\mathbf{s}_i$ neighbors, inside which particle i has direct information links only with its neighbors, its velocity updating formula is dependent on $\mathbf{v}^i(t)$, $\mathbf{p}^i(t)$ and $\mathbf{b}^i(t)$ as well as $\mathbf{x}^i(t)$. If $\mathbb{S}_i$ is expanding and involving more particles being i's neighbours, it will eventually contain all the individuals of the swarm, i.e. $\mathbb{S}_i$ emerges to $\mathbb{S}$ and $\mathbf{b}^i(t)$ coincides with one point $\mathbf{b}(t)$. Thus the velocity update formula for *Gbest* model can be expressed as:

$$\mathbf{v}^i(t+1) = \mathbf{v}^i(t) + C_1 R_1(\mathbf{p}^i(t) - \mathbf{x}^i(t)) + C_2 R_2(\mathbf{b}(t) - \mathbf{x}^i(t)) \tag{2.29}$$

In summary, Global and Local refer to the scope of a particle i's sub-domain $\mathbb{S}_i$.

2.4 Drawbacks of PSO

As a member of stochastic search algorithms, PSO has two major drawbacks [85]. The first drawback of PSO is its premature character, i.e. it could converge to local minimum. According to Angeline [8], although PSO converges to an optimum much faster than other evolutionary algorithms, it usually cannot improve the quality of the solutions as the number of iterations is increased. PSO usually suffers from premature convergence when high



(a) Von Neumann topology

(b) Ring topology

(c) Wheel topology

(d) Gbest topology

Figure 2.3: Common topologies of PSO

multi-modal problems are being optimized. The main reason is that for the standard PSO (especially *Gbest* PSO), particles converge to a single point which is on the line connecting the global best and the personal best positions. Nevertheless this point is not guaranteed to be a local optimum and may be called equilibrium point, which has been proven by Van den Bergh [35]. Another reason for a premature problem can be seen that the high rate of information flow between particles during optimization can result in causing a swarm to consist of similar particles (a loss in diversity). This increases the possibility of being trapped in local optima [109]. Several efforts have been made to advance a variation of PSO which addresses this problem satisfactorily. Some of them have already been discussed, including inertia weight, the constriction factor and so on. Further modifications are discussed in the next section.

The second drawback is that the PSO has a problem-dependent performance. This dependency is usually caused by the way parameters are set, i.e. assigning different param-

eter settings to PSO will result in high performance variance. In general, based on the *no free lunch theorem*, no single parameter setting exists which can be applied to all problems and performs dominantly better than other parameter settings. For instance, increasing the value of inertia weight, ω will increase the speed of the particles resulting in more exploration ability (global search) and less exploitation (local search), i.e. PSO with a higher ω can better find solutions of multi-modal problems and PSO with a lower ω can find optimum faster for uni-modal problems. Thus obtaining the most proper value of ω is not an easy task and it may differ from one problem to another. There are now two common ways to deal with this problem. One way is to use self-adaptive parameters. Self-adaptation has been successfully applied to PSO by Clerc [28], Shi and Eberhart [121], Hu and Eberhart [65], Ratnaweera et al. [104] and Tsou and MacNish [130] and so on. Another solution is to use PSO hybridized with another kind of optimization algorithm, so that the PSO can benefit from the advantages of another approach. Hybridization has been successfully applied to PSO by Angeline [9], Løvberg [86], Zhang and Xie [142]. All improvements to PSO that have diminished the impact of the two aforementioned disadvantages will be discussed in detail in the next section.

2.5 Current variants of the PSO

After the PSO was discovered, it attracted the attention of many researchers for its beneficial features. Various researchers have analyzed it and experimented with it, including mathematicians, engineers, physicists, biochemists, and psychologists and many variations were created to further improve the performance of PSO. In the process, a certain body of lore emerges as the algorithm is well understood, as the theorists argue the inbeing of adaptive systems, and as programmer's trails and errors result in obvious improvements. This section showcases some of the main focal points of current research. It also offers some suggestions on how to possibly improve the PSO further.

2.5.1 Introduction to the variants of the PSO

Let us think about the PSO in a different way. After several iterations, the particle's current individual best position will converge to the global best position, so that the second and third parts of formula (2.15) tend to be zero. A potential danger for this algorithm is the fact that the process of the update only relates to the momentum part, or to say, particle moves following its old trajectory, which is a potential risk for this algorithm. If the global best position is close to a local minimum, PSO will converge to that point. To be precise, this means that it cannot be guaranteed that the result is locally minimal,it merely means that all the particles converge to a global best position (i.e. equilibrium point) that has been discovered up to this point in time by the whole swarm. So far there are numerous variants of PSO that deal with this disadvantage. All of the variants can be divided into five categories:

1. Variants based on the modifications of the original PSO (seen in class I in table 2.1), discussed in subsection 2.5.2.

2. Variants inspired by some other algorithms (especially evolutionary algorithms) (seen in class II in table 2.1), discussed in subsection 2.5.3.

3. Hybrid variants (i.e. these variants combine PSO and another kinds of optimization algorithms.) (seen in class III in table 2.1), discussed in subsection 2.5.4.

4. Variants for solving integer programming (seen in class IV in table 2.1), discussed in subsection 2.5.5.

5. Other variants (seen in class V in table 2.1), discussed in subsection 2.5.6.

It is important to realize that in the face of the huge amount of variants, their designation into categories is very difficult and is dependent on the level of knowledge of the individual researcher, his research field, his manner of thinking, as well as the number of acquired papers - to name but a few. Although the chosen classification in this section is the product of thorough reflection on the author's part, it is by no means the only possible point of view. It is hoped that it can at least be the basis for a transparent investigation of the variants of the PSO.

Context		Full name	Brief Introduction	Reference
I	GCPSO	Guaranteed Convergence PSO	Induce a new particle searching around the global best position found so far.	[131]
	MPSO	Multi-start PSO	Use GCPSO recursively until some stopping criteria is met.	[35]
	PSOPC	PSO with Passive Congregation	Add a passive congregation part to the particle's velocity update formula.	[60]
	Selecting	Selecting strategy	Utilize new seleting on strategies to get *pbest* and *gbest*.	[100]
	FIPSO	Full informed PSO	Use all particle i's neighbors' best personal position to update i's velocity.	[89]
	SPSO	Species-based PSO	Use several adaptively updated species-based sub-swams to search design space.	[80]
	APSO	Adaptive PSO	Impove swarm's local and global searching ability by inserting self-organization theory.	[137]
	CPSO	Clan PSO	Use sub-swarms to search design space, sub-swams communicate with each other every some iterations.	[25]
	SPSO	Guaranteed Global Convergence PSO	Particle i's position will be regenerated randomly if it is too close to the *gbest*.	[31]
	PSO-DT	PSO with Disturbance Term	Induce a disturbance term to the velocity update formula.	[59]
	CPSO	Cooperative PSO	Use multi-swarms to search different dimensions of the design space by employing cooperative behaviour.	[132]
	Selection	Selection	Particles are sorted based on their performance, the worst half is then replaced by the best half.	[9]
II	DPSO	Dissipative PSO	Add mutation to the PSO in order to prevent premature convergence.	[138]

Continued on next page

continued from previous page				
Context		Full name	Brief Introduction	Reference
	NPSO	Niche PSO	Induce niche theory to the ability to handle more complex optimization problems.	[24]
III	HPSO	Hybrid of GA and PSO	Utilize the mechanism of PSO and the selection mechanism of GA.	[41]
	EPSO	Hybrid of EP and PSO	Incorporate a selection procedure to the PSO, as well as self-adapting properties for its parameters.	[92]
	PSACO	Hybrid of PSO and ACO	Use ACO to improve the particles' positions after they move to new positions.	[71]
	PDPSO	Preserving Diversity in PSO	Add memory capacity to each particle in a PSO algorithm to maintain diversity of the whole swarm.	[61]
IV	BPSO	Binary PSO	Discrete design variables are expressed in binary form in this variant.	[73]
	RPSO	Rounding-off PSO	Each is rounded off to its nearest integer in order to compute its fitness.	[76]
V	ALPSO	Augmented lagrangian PSO	Use PSO to update factors used by augmented lagrangian algorithm by solving sub-problems.	[119]

Table 2.1: A brief overview of PSO variants discussed in section 2.5

2.5.2 Variants based on the modifications of the original PSO

In this subsection such kind of variants is discussed here: the ideas behind these modifications are first implemented to the PSO. The PSO uses few parameters to investigate complex environments, so the performance of the PSO is sensitive to the selection of these parameters, some numerical experiments have partly validated that. So far there doesn't exist a comprehensive convergence proof for PSO. For this reason, it is very difficult to set parameters theoretically, although some researchers have issued guidelines concerning this task. These have already been discussed in the last section. Some other researchers invent creative concepts and apply them to boost the performance of the PSO. Nice results from benchmark tests have confirmed these modifications.

2.5.2.1 GCPSO: Guaranteed Convergence PSO

The basic idea of GCPSO [35] is to introduce an additional particle, which searches the region around the current global best position, i.e. its local best position is equal to the current

global best position. In that manner, the current global best particle is treated also as a member of the swarm(e.g. particle τ), the update formula for this particle is seen below:

$$\mathbf{v}^{\tau}(t+1) = -\mathbf{x}^{\tau}(t) + \mathbf{b}(t) + \omega\mathbf{v}^{\tau}(t) + \rho(t)(1-2\mathbf{r}) \tag{2.30}$$

It is noted that this variant is so far only applied to the *Gbest* mode.The other particles in the swarm continue to use the normal velocity update formula, e.g. formula (2.15). Here the term $-\mathbf{x}^{\tau}(t) + \mathbf{b}(t)$ looks like the global cognitive part in formula (2.15). Because the global best position and the individual best position are coincident. $\rho(t)(1-2\mathbf{r})$ substitutes the "social" part of the formula (2.15) to increase its search ability, which causes the additional particle to perform a random search in an area surrounding the global best position. Here $\mathbf{r}$ is a vector randomly generated in the domain $[0,1]^n$ and $\rho(t)$ is the diameter of the search area and dynamically adapted based on the behaviour of the swarm, i.e. if the swarm always finds a better position than the current global best position in consecutive iterations, the search diameter will become larger; if the swarm always fails to find a better position than the current global best position in consecutive iterations, the search diameter will become smaller. The update formula of the diameter is as follows:

$$\rho(t+1) = \begin{cases} 2\rho(t) & \text{if \#successes} > s_c \\ 0.5\rho(t) & \text{if \#failures} > f_c \\ \rho(t) & \text{otherwise} \end{cases} \tag{2.31}$$

Where terms #successes and #failures are defined as the number of the consecutive successes or failures, respectively, and the definition of failure is . The threshold parameters s_c and f_c are defined empirically. Since in a high dimensional search space, it is difficult to obtain a better value in only a few iterations, thus recommended values are thus $s_c = 15$ and $f_c = 5$. On some benchmark tests, the GCPSO has shown a nice performance of locating the minimal of a uni-modal function with only a small amount of particles. This implies that it does not need many particles to facilitate a local search. Compared with the original PSO, it also has a faster convergence on uni-modal functions. This approach is a good supplement to the original PSO, however it has a small flaw, i.e. sometimes if the current global best position is located on a local minimum which is not near the global minimum, the additional particle could possibly fail to find any better position around this local minimum and the algorithm would converge to the local minimum eventually. Van Den Bergh later proved that it is a locally convergent particle swarm optimizer [131]. However, since this approach has a very strong local search ability and an easy form, Van den Bergh has offered some extensions to the GCPSO. Some variants will be discussed in the next section, while the main concepts of these variants are inspired by other algorithms and the GCPSO is only used to solve sub-problems.

2.5.2.2 MPSO: Multi-start PSO

Van den Bergh [35] proposed MPSO which is an extension to GCPSO in order to turn it into a global search algorithm. MPSO works as follows:

1. Initialize all the particles in the design space randomly.

2. Apply the GCPSO until it converges to a local optimum. Save the position of this local optimum.

3. Repeat Steps 1 and 2 until some stopping criteria are satisfied.

In Step 2, the GCPSO can be replaced by the original PSO. Several versions of MPSO were proposed by Van den Bergh [35] based on the criterion used to determine the convergence of GCPSO. One approach is to measure the rate of change in the objective function as follows:

$$f_{ratio} = \frac{f(\mathbf{b}(t+1)) - f(\mathbf{b}(t))}{f(\mathbf{b}(t))} \tag{2.32}$$

If f_{ratio} is less than a user-specified threshold (this value is depend on the range of the objective function values, as well as the computer precision) then a counter is incremented. The GCPSO is assumed to converge to a minimum if the counter reaches a certain limit. According to Van den Bergh [35], MPSO generally performed better than GCPSO in most of the tested cases. Theoretically, if the swarm could be re-generated an infinite amount of times, the GCPSO could eventually find the global optimal. However, the performance of MPSO obviously degrades when the number of design variables in the objective function increases.

2.5.2.3 PSOPC: PSO with Passive Congregation

This approach [60] creates an additional part at the end of the velocity update formula (2.15) known as passive congregation part. The basic idea is that individuals need to monitor both their environment and their surroundings. Thus, each group member receives a multitude of information from other members which may decrease the possibility of a failed attempt at detection or a meaningless search. This kind of information exchange can be realized by a model called passive congregation. The PSOPC is defined as:

$$\mathbf{v}^i(t+1) = \omega\mathbf{v}^i(t) + C_1R_1(\mathbf{p}^i(t) - \mathbf{x}^i(t)) + C_2R_2(\mathbf{b}^i(t) - \mathbf{x}^it) + C_3R_3(\mathbf{x}^r(t) - \mathbf{x}^i(t)) \tag{2.33}$$

where r is a random integer selected from $[1, s]$. It must be noted that each particle obtains passively additional information from another particle that is selected at random. This could increase the diversity of the swarm and lead to a better result. This approach was tested with a benchmark test and compared with standard *Gbest* mode PSO, *Lbest* mode PSO and PSO with a constriction factor, respectively [60]. Experimental results indicate that the PSO with passive congregation improves the search performance on the benchmark functions significantly.

2.5.2.4 Selecting strategy

Padhye [100] has collected strategies on how to select *gbest* and *pbest*. These strategies were originally designed for PSO to solve multi-objective problems with a good convergence, as well as a diversity and spread along the Pareto-optimal front. However they can also be

extended to other kinds of optimization problems. Typical strategies to select the *gbest* make use of random selection, choosing a particle that dominates many particles or the sigma method. But, how to select the personal best has not been studied thoroughly so far. In past studies a particle i is allowed to remember its best position $\mathbf{p}^i$ only. The author brings forward some new ideas about how to select *gbest* and *pbest*, together with the existing issues which are described bellow:

1. **Random**: This is a simple strategy that each particle selects randomly a non-dominated member (or position) from the global and personal archive as its *gbest* and *pbest* respectively.
2. **Wtd.**: In this approach, in order to keep swarm diversity, a higher weight is allotted to those criteria in which the particle is already good and a weighted sum is calculated. Corresponding members in global and personal archives which have the highest weighted sums are chosen [23]. This strategy hopes to promote to improve a particle's position where its objective function value is already good.
3. **Newest**: In this approach the personal best is updated only when a new non-dominated position is reached. So the particle will not be drawn back to previously searched regions. Hence, a better diversity of solutions is expected.This method is only applied in selecting the personal best.
4. **Indicator-based**: This is a newly proposed approach in which such a *gbest* or *pbest* is chosen which contributes most to the hyper volume with respect to a reference point. Usually, the individual itself is chosen as the reference point. This strategy hopes to promote to increase the diversity in middle parts of the Pareto front, but shows a poor performance at the extreme ends.
5. **Dominance based probability**: Among the archive members which dominate the individual, the selecting guides are based on a probability distribution. Archive members which dominate a greater number of individuals are assigned a higher probability of getting selected. In the past, such a strategy [7] has been successfully applied to the selection of global guides and its extension for selecting *pbest* is also made.
6. **Sigma-Sanaz**: The Sigma method was originally proposed [95] for selecting *gbest*. The idea behind this strategy is to allow the individuals to be attracted to the non-dominated members which are closest to them.

Based on these strategies, new universal modifications may be achieved in the future.

2.5.2.5 FIPSO: Full informed PSO

It is an alternative that is conceptually more concise and promises to perform more effectively than the traditional particle swarm algorithm. In this new version, the particle uses information from all its neighbors, rather than just the best one. This approach uses a dense velocity update formula that can be described as:

$$\mathbf{v}^i(t+1) = \chi(\mathbf{v}^i(t) + \varphi(\hat{\mathbf{p}}^i(t) - \mathbf{x}^i(t))) \tag{2.34}$$

where χ is the constriction factor which has already been discussed before, $\varphi = C1R1 + C2R2$ and $\hat{\mathbf{p}}^i(t)$ is calculated as $\hat{\mathbf{p}}^i(t) = \frac{C1R1\mathbf{b}^i(t) + C2R2\mathbf{p}^i(t)}{C1R1 + C2R2}$. Then this approach is described as:

$$\varphi_k = \mathbf{U}\Big[0, \frac{\varphi_{max}}{\mid \mathbf{s}_i \mid}\Big], \quad k \in \mathbb{S}_i \tag{2.35}$$

$$\hat{\mathbf{p}}^i = \frac{\sum_{k\in\mathbb{S}_i} w_k \varphi_k \mathbf{p}^k}{\sum_{k\in\mathbb{S}_i} w_k \varphi_k} \tag{2.36}$$

where $\mathbf{s}^i$ is the number of particle i's neighbors based on the information topology and $\mathbf{p}^k$ denotes the best position discovered by particle k so far. $\mathbf{w}$ is a weight factor describing the contribution of particle k hypothesized to be relevant. In this paper [89], several strategies were advanced by the authors, which are listed below:

1. **FIPS**: the fully informed particle swarm with a constant W for all particles, i.e., where all contributions have the same value;
2. w**FIPS**:a fully informed swarm, where the contribution of each neighbor is weighted by the worth of its previous best;
3. wd**FIPS**: also fully informed, with the contribution of each neighbor weighted by its distance in the search space from the target particle;
4. Self: a fully informed model, where the particles own previous best received half the weight;
5. wself: a fully informed model, where the particles own previous best received half the weight and the contribution of each neighbor was weighted by the worth of its previous best.

These strategies were tested with five types of topologies in order to achieve a detailed conclusion. Finally the authors [89] gave some advanced suggestions: the w**FIPS** with u-topology passed all the tests and gave rise to the best results best results whilst being relatively slow. If speed is a requirement, its relative slowness may create problems, then the unweighed FIPS with us-square was suggested, which held a good balance between successful rate and speed in all the tests.

Several variants are based on this approach,such as: PSOOP(Particle Swarm Optimization with Opposite Particles) [134], TSPSO(two-stage particle swarm optimizer) [144], RDNPSO(Random Dynamic Neighborhood PSO) [93], RDNEMPSO(Randomized directed neighborhoods with edge migration in particle swarm optimization) [94], P$\mathbf{S}^2$O(Particle Swarms Swarm Optimizer) [26]

2.5.2.6 SPSO: Species-based PSO

This approach [80] divides the swarm population into species-specific sub-populations based on their similarity which is measured in Euclidean distance from the center of a specie

to its boundary. So, any particle falling into this circle can be classified as a member of these species, the center of these species is the position of a dominated particle. At each step, the species seeds are selected as neighborhood bests for the species groups. After successive iterations, these subpopulations could find multiple local optima from which the global optimum could be identified. This approach was tested on a series of widely used multi-modal test functions and found all the global optima for the all test functions with good accuracy.

An extension of SPSO was proposed in [81]and it took on board some useful concepts, such as quantum swarms and so on. The results from the moving peaks benchmark test functions show that the extension could greatly increase adaptability to find the optima in dynamic environments.

2.5.2.7 APSO: Adaptive PSO

This approach [137] is based on the self-organization theory. As the evolution of swarm unfolds, the particles may loss the local and global search abilities when their individual best positions are very close to the best positions achieved from their neighbours which are called inactive particles. This problem has already mentioned in GCPSO. To overcome this issue, the following improvements have been made by the author:

1. Define in advance a constant ε as a measurement to detect inactive particles.
2. The positions and velocities of the particles will be randomly regenerated.

There are also some other adaptive variants of PSO, such as: QPSO(quantum-behaved particle swarm optimization) [125], APSO(Adaptive particle swarm optimization using velocity information of swarm) [139].

2.5.2.8 CPSO: Clan PSO

This approach [25] includes the following steps:

1. Create clan topologies: Clans are groups of individuals, or tribes, united by a kinship based on a lineage or a mere symbolic. For each iteration, each clan performs a search and marks the particle that had reached the best position of the entire clan.
2. Leader delegation: The process of marking the best within the clan is exactly the same as stipulating a clans symbol as guide. It is the same as delegating the power to lead the others in the group.
3. Leaders' conference: Leaders of all the clans exchange their informations using *Gbest* or *Lbes*t models.
4. Clans feedback information: After the leaders conference, each leader will return to its own clan. The new information acquired in the conference will be spread widely within the clan.

Results have shown that the proposed topology achieves better degrees of convergence. This approach could also be combined with other PSO approaches to generate different ways to search and optimize.

2.5.2.9 SPSO: A Guaranteed Global Convergence PSO

SPSO is proposed by Cui and Zeng [31]. In this approach, if the global best position is replaced by a particle's position in some interaction, this particles' position will be regenerated and if a particles' new position coincides with the global best position, its position will also be regenerated randomly. The authors has proved that this is a guaranteed global convergence optimizer and through some numerical tests this optimizer showed its good performance.

2.5.2.10 PSO-DT: PSO with Disturbance Term

This approach [59] introduces a disturbance term to the velocity update formula(). The new updating formula can be defined as:

$$\mathbf{v}^i(t+1) = \mathbf{v}^i(t) + C_1 R_1(\mathbf{p}^i(t) - \mathbf{x}^i(t)) + C_2 R_2(\mathbf{b}^i(t) - \mathbf{x}^i(t)) + \alpha(R_3 - 0.5) \tag{2.37}$$

Where α is a small constant, R_3 is from a random distribution in the $(0,1)$ range, the others parameters is the same as standard PSO. In the early calculation, because α is very small compared to the other three terms, and the mean value of $(R_3 - 0.5)$ is equal to zero, the disturbance term can even be ignored since it exerts little impact on the updating and searching capability of the whole optimization. In the middle and later phases the velocity continually decreases and the disturbance term ensures that the searching velocity of the particles will not drop down to zero. That way, the optimization will not stagnate and the update can continue. This is a way of overcoming the drawbacks associated with falling into local optima when dealing with the standard PSO. In this way, the achievement of more exact solutions is made possible.

2.5.2.11 CPSO: Cooperative PSO

Van Den Bergh and Engelbrecht [132] proposed a modified Particle Swarm Optimizer named CPSO. The CPSO could significantly improve the performance of the original PSO by utilizing multiple swarms for optimizing different components of the solution vector by employing cooperative behaviour. Firstly the search space is partitioned by dividing the solution vectors into smaller vectors, based on the partition several swarms will be randomly generated in different parts of the search space and used to optimize different parts of the solution vector. Two cooperative PSO models are proposed. One of them, known as CPSO-$\mathbf{S}_k$ is a direct extension of Potters cooperative coevolutionary genetic algorithm (CCGA) to the standard PSO. A swarm with an n-dimensional vector is divided into n swarms of one-dimensional vectors and each swarm attempts to optimize a single component of the solution vector. The other variant is known as CPSO-$\mathbf{H}_k$ and is a two-stage optimizer. Each

iteration is made up of two processes: First, CPSO-$\mathbf{S}_k$ is implemented to obtain a sequence of potential solution points. Next, half of those are randomly selected for an update with the original PSO. In case the new position dominates the previous one, the information of the sequence will be updated.

2.5.2.12 Selection

In this approach [9], each particle is ranked based on a comparison of its performance compared with that of a group of randomly selected particles. A particle is awarded one point whenever it shows better fitness in a tournament than another. Members of the population are then organized in descending order according to their accumulated points. The bottom half of the population is then replaced by the top half. This step reduces the diversity of the population. The results show that the hybrid approach performed better than the PSO (with and without ω) in uni-modal functions. However, the hybrid approach performed worse than the PSO for functions with many local optima. Therefore, it can be concluded that although the use of a selection method improves the exploitation capability of the PSO, it reduces its exploration capability.

2.5.3 Variants inspired by evolutionary algorithms

This subsection deals with variants whose logic is derived from other algorithms and that is now successfully applied to the PSO.

2.5.3.1 DPSO: Dissipative PSO

DPSO was first proposed by Xie et al. [138] to add random mutation to PSO in order to prevent premature convergence. This could be thought of as an inspiration for GA. DPSO introduces negative entropy through the addition of randomness to the particles (after executing equation (2.15) and (2.16)) as follows:

If $(r1_i(t) < c_v)$, then $\mathbf{v}^i(t+1) = r2(t)\mathbf{v}_{max}$

If $(r3_i(t) < c_l)$, then $\mathbf{x}^i(t+1) = R(t)$

where $r1_i(t) \sim U(0,1)$, $r2(t) \sim U(0,1)$ and $r3_i(t) \sim U(0,1)$; c_v and c_l are chaotic factors in the range [0,1] and $R(t) \sim U(x_{min}, x_{max})$ where x_{min} and x_{max} are the lower and upper bound of the search space. Note that if c_v and c_l are nearly equal to one, this approach is like a purely random search algorithm, whereas if they are closer to zero, this approach is like standard PSO. The application of chaos from the environment forces the system to move in a state far from equilibrium. Then, the self-organization that is a characteristic of a dissipative structure comes into being. This consists of non-linear interactions in the swarm, leading to "sustainable development" because of the fluctuations. The results showed that DPSO performed better than standard PSO when applied to the benchmark problems. The concept of mutation is also adopted by other variants, such as: PSOG(PSO with gaussian mutation) [63], PSOM(PSO with mutation) [123].

2.5.3.2 NPSO: Niche PSO

In ecology, a niche is a term that describes the relational position of a species or population within its ecosystem. In succinct terms, a niche shows how an organism makes its living. Applied to optimization algorithms, a niche can be viewed as a small population which constitutes the whole population together with other niches. This concept has already been used in some modifications of Genetic algorithms in order to shape the search ability in case of multi-model problems.

PSO and its variants have been shown to effectively solve uni-modal optimization problems. These PSO algorithms are not, however, well suited for solving multiple problems because of the way in which they socially exchange information regarding a good global solution. So this concept was first induced to PSO in [24] in order to heighten its ability to handle more complex optimization problems that can search for multiple solutions in parallel. The workflow of this approach is shown below:

1. Initialize main particle swarm.
2. Train main swarm particles using one iteration of the individual memory only model.
3. Update fitness of each main swarm particle.
4. For each sub-swarm: use GCPSO to update each particle's position and then update swarm radius
5. If possible, merge sub-swarms
6. Allow sub-swarms to absorb any particles from the main swarm that moved into it.
7. Search main swarm for any particle that meets the partitioning criteria If any is found create a new sub-swarm with this particle and its closest neighbor.
8. Repeat from 2 until stopping criteria are met.

The above description shows that the most important focus for variants of this nature is creating sub-swarms and to absorb corresponding particles from the main swarm. The author applied the following criteria: If a particle's fitness shows very little change over a small number of iterations of the learning algorithm, a sub-swarm is then created with the particle and its closest topological neighbor measured by the Euclidean distance. Experimental results showed that this algorithm successfully located all maxima for all the simulation runs.

The niche concept is also applied in many variants of PSO, such as ASNPSO (adaptive sequential niche particle swarm optimization) [141], PVPSO (parallel vector-based particle swarm optimizer) [116]. There are also some enhancements for niche based PSO such as Enhancing the NichePSO [42], Adaptively choosing niching parameters in a PSO [19].

2.5.4 Hybrid variants

A natural evolution of the particle swarm algorithm can be achieved by incorporating other evolutionary computation techniques. Many authors have considered incorporating selection, mutation and crossover, as well as the differential evolution (DE) into the PSO algorithm. The main goal is to increase the diversity of the population by either preventing each particle to move close to other particles or using the other evolutionary algorithm as a sub-step to improve the particle's position. The most common ones are discussed bellow.

2.5.4.1 HPSO: Hybrid of Genetic Algorithm and PSO (GA-PSO):

HPSO utilizes the mechanism of PSO and a natural selection mechanism which is usually utilized by EC such as the employment of genetic algorithms (GA). Since the search procedure by PSO deeply depends on *pbest* and *gbest*, the searching area is limited by *pbest* and *gbest*. On the other hand, by introducing a natural selection mechanism, the limiting effects of *pbest* and *gbest* can be eroded so that a greater search area can be achieved. Agent positions with low evaluation values are replaced by those with high evaluation values using the selection. On the contrary, *pbest* information of each agent is maintained. Therefore, intensive search in a current effective area and dependence on the past high evaluation position are realized.

The GA-PSO algorithm basically employs a major aspect of the classical GA approach, which is the capability of "breeding." El-Dib et al. [41] considered the application of a reproduction system that modifies both the position and velocity vectors of randomly selected particles in order to further improve the potential of PSO to reach an optimum.

$$\mathbf{child}^1(x) = p \cdot \mathbf{parent}^1(x) + (1-p) \cdot \mathbf{parent}^2(x) \tag{2.38}$$

$$\mathbf{child}^1(v) = (\mathbf{parent}^1(v) + \mathbf{parent}^2(v)) \frac{\parallel \mathbf{parent}^1(v) \parallel}{\parallel \mathbf{parent}^1(v) + \mathbf{parent}^2(v) \parallel} \tag{2.39}$$

$$\mathbf{child}^2(x) = p \cdot \mathbf{parent}^2(x) + (1-p) \cdot \mathbf{parent}^1(x) \tag{2.40}$$

$$\mathbf{child}^2(v) = (\mathbf{parent}^1(v) + \mathbf{parent}^2(v)) \frac{\parallel \mathbf{parent}^2(v) \parallel}{\parallel \mathbf{parent}^1(v) + \mathbf{parent}^2(v) \parallel} \tag{2.41}$$

where $\mathbf{parent}^1(x)$ and $\mathbf{parent}^2(x)$ represent the position vector of two randomly selected particles, $\mathbf{parent}^1(v)$ and $\mathbf{parent}^2(v)$ are their corresponding velocities, $\mathbf{child}^1(x)$ and $\mathbf{child}^2(x)$ are the offspring of the breeding process.

2.5.4.2 EPSO: Hybrid of Evolutionary Programming and PSO

Evolutionary PSO incorporates a selection procedure into the original PSO algorithm, as well as self-adapting properties for its parameters. Miranda and Fonseca proposed adding the tournament selection method used in evolutionary programming (EP) for this purpose [92]. In this approach, the update formulae remain the same as in the original PSO algorithm; however, the particles are selected as follows.

The fitness value of each particle is compared with other particles and scores a point for each particle with a worse fitness value. Based on this, the members of a population are ranked. The current positions and velocities of the best half of the swarm replace the positions and velocities of the worse half. The individual best of each particle of the swarm (best and worst half) remain unmodified. Therefore, at each iteration step half of the individuals are moved to positions of the search space that are closer to the optimal solution than their previous positions while keeping their personal best points.

The difference between this method and the original particle swarm is that the exploitative search mechanism is used in a more pronounced way. This should help the optimum to be found more consistently than the original particle swarm. The general EPSO scheme can be summarized as follows:

- ◇ Replication: Each particle is replicated.
- ◇ Mutation: Each particle has its weight mutated.
- ◇ Reproduction: Each mutated particle generates an offspring according to the particle movement rule.
- ◇ Evaluation: Each offspring has a fitness value.
- ◇ Selection: Stochastic tournament is carried out in order to select the best particle which survives until the next generation.

2.5.4.3 PSACO: Hybrids of PSO and ACO

This approach has two stages [71]. First, a swarm is randomly generated in the design domain. After that ant colony optimization is applied to improve the particles' positions. The improved particles proceed to the next update step described in formula (2.18) and (2.16) until some stopping criterion is met.

2.5.4.4 PDPSO: Preserving Diversity in PSO

The approach [61] adds memory capacity to each particle in a PSO algorithm to maintain spread and, therefore, diversity by providing individual specific alternate target points to be used at times instead of the current local best position $\mathbf{p}^i$, which is conceptually derived from the pheromone trails of the ACO algorithm. To optimize this effect each particle in the swarm maintains its own memory. The maximum size of the memory and the probability that one of the points will be used instead of the current local best point $\mathbf{p}^i$ in formula (2.15) are user specified parameters. The current particle i's position $\mathbf{x}^i$ will be added to its memory if the fitness of this point is better than that of the least fit point stored. It may also be required to differ by at least a specified amount from any point already in the memory. The new memory point replaces the least fit point if the memory is full. When a point from a particle's memory is to be used, the point may be chosen randomly or the probability of selection may be fitness based (with better fitness producing a higher probability of selection).

2.5.4.5 HSPSO: Hybrid of Simplex algorithm and PSO

In this variant [45], initially $3n+1$ particles will be randomly generated. After an evaluation of the fitness of each particle, the particle will be ranked based on its fitness value. The first $n+1$ particles are updated using a simplex search algorithm, and the rest $2n$ particles are updated using the standard PSO.

2.5.5 Variants for solving integer programming

2.5.5.1 BPSO: Binary PSO

Binary PSO was first proposed by Kennedy and Eberhart [73]. In this approach, the position in design space of each particle is expressed in a binary string. Each component (i.e. X_j^i) of a string has only two status, YES/TRUE=1, and, NO/FALSE=0. In that sense, the velocity (i.e. v_j^i)update formula (2.15) keeps its original form, however, the position update formula (2.16) has to be changed as

$$\begin{aligned} &\text{if } (rand() < S(v_j^i)\ ) \text{ , then } X_j^i = 1; \\ &\text{else, } X_j^i = 0 \end{aligned} \tag{2.42}$$

where $rand()$ is random number selected from a uniform distribution in $[0.0, 1.0]$ and $S()$ is a sigmoid limiting transformation and given by

$$S(v_j^i) = \frac{1}{1+\exp(-v_j^i)}$$

Note that $S()$ can be seed as the mutation rate of X_j^i with given velocity v_j^i and the ultimate mutation rate is controlled by v_{max}, higher or lower value of v_{max} can cause the threshold being too close to 0.0 or 1.0, normal choice is $v_{max} = 4$, therefore, mutation rate is limited into a proper interval $[0.018, 0.982]$. Of course this approach can also be applied to continuous optimization problems, since any continuous real value can also be represented as a bit string.

2.5.5.2 RPSO: Rounding-off PSO

In a more general case, when integer solutions (not necessarily 0 or 1) are needed, the optimal solution can be determined by rounding off the real optimum values to the nearest integer, i.e. the optimal solution can be determined by rounding off the real optimum values to the nearest integer. This is first proposed by Laskar et el. [76]. In this approach, PSO searches the real number space to determine the new position of each particle $x_k^i(t) \in \mathbb{R}$. Once $x^i(t) \in \mathbb{R}$ is obtained, its value in the kth dimension is rounded to the nearest integer value using the bracket function

$$X_k^i(t) = [x_k^i(t)], \quad X_k^i(t) \in \mathbb{Z}, \quad x_k^i(t) \in \mathbb{R} \tag{2.43}$$

The authors have tested three variants of PSO (PSO only with weight inertia, PSO only with constriction factor and PSO with both weight inertia and constriction factor) and also compared the results with the Branch-and-Bound method. The results presented in [76] using integer PSO show that the performance of the method is not affected by truncating the real values of the particles. Furthermore, integer PSO has a high success rate in solving integer programming problems even when other methods, such as, branch-and-bound fails. In this work, LPSO are applied to large scale discrete truss topology optimization using the aforementioned rounding strategy to handle integer design variable.

2.5.6 Others

2.5.6.1 ALPSO: Augmented Lagrangian PSO

This approach was first proposed by Sedlaczek and Eberhard in [119]. In general, the Augmented Lagrangian is a method of transforming constrained optimization problems into unconstrained form. It can be expressed in the following way

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^{n_g+n_e} \lambda_i \theta_i(\mathbf{x}) + \sum_{i=1}^{n_g+n_e} r_i \theta_i^2(\mathbf{x}) \tag{2.44}$$

with

$$\theta_i = \begin{cases} g_i(\mathbf{x}), & i \in [1, \cdots, n_g] \\ \max\left[h_{i-n_g}(\mathbf{x})\right], & i \in [n_g+1, \cdots, n_g+n_i] \end{cases} \tag{2.45}$$

In this approach, the factors $\boldsymbol{\lambda}$ and $\mathbf{r}$ can be updated by

$$\lambda_i(t+1) = \lambda_i(t+1) + 2r_i\theta_i(\mathbf{x}(t)), \tag{2.46}$$

$$r_i(t+1) = \begin{cases} 2r_i(t) & \text{if} \quad \mid g_i(\mathbf{x}(t)) \mid > \mid g_i(\mathbf{x}(t-1)) \mid \wedge \mid g_i(\mathbf{x}(t)) \mid > \epsilon_g \\ \frac{1}{2}r_i(t) & \text{if} \quad \mid g_i(\mathbf{x}(t)) \mid \leq \epsilon_g \\ r_i(t) & \text{else} \end{cases} \tag{2.47}$$

$$\text{with} \quad i \in \{1, \ldots, n_g\} \tag{2.48}$$

and

$$r_{i+n_g}(t+1) = \begin{cases} 2r_{i+n_g}(t) & \text{if} \quad \mid h_i(\mathbf{x}(t)) \mid > \mid h_i(\mathbf{x}(t-1)) \mid \wedge \mid h_i(\mathbf{x}(t)) \mid > \epsilon_h \\ \frac{1}{2}r_{i+n_g}(t) & \text{if} \quad \mid h_i(\mathbf{x}(t)) \mid \leq \epsilon_h \\ r_{i+n_g}(t) & \text{else} \end{cases} \tag{2.49}$$

$$\text{with} \quad i \in \{1, \ldots, n_e\} \tag{2.50}$$

where ϵ_g and ϵ_h are the user-defined tolerances for acceptable constraint violations and $\mathbf{x}(t)$ is the solution of problem (2.44) sovled by standard PSO within a limited number of iterations. Note that these updating schemes are derived from the stationarity condition of (2.44). This process continues until the stopping criterion is met.

2.6 Application fields of PSO

Generally, the PSO algorithm has the following advantages compared with other optimization algorithms:

1. It is a simple algorithm with only a few parameters to be adjusted during the optimization process, so that it is easy to be implemented with any modern computer language.
2. It is a powerful tool, because there are no application limits to it, almost all kinds of optimization problem can be solved by PSO, normally in the original form.
3. It has a superior convergence speed compared with other evolutionary algorithms, i.e. some optimization problems can be solved more rapidly by PSO than by other evolutionary algorithms.

Due to its aforementioned attractive features, it has gained a lot of attention in recent years and is applied in many fields. Poli [102] did a general search in the IEEE Xplore database and found a list of about 1100 papers matching the keyword "Particle Swarm Optimization". Roughly 300 of them are concerned with the improvement of the PSO and the rest deals with the application of the PSO in 26 different categories. Because the IEEE Xplore database focuses on electrical engineering, the mentioned categories do not entail all application fields of PSO. Some other application fields could be added to this list, e.g. structural optimization in sizing, shape and topology, robust design, etc. It appears that PSO is used to solve traditional optimization problems (e.g. structural optimization design, antenna design, etc.), as well as new optimization problems (e.g. image and video analysis applications, signal processing, applications in electronics and electromagnetic, security and military applications, etc.) From the above survey, it can be seen that the application field of PSO is now very wide. It is likely to cover an even broader are of optimization as research efforts continue.

Chapter 3

The modified Particle Swarm Optimization

Although PSO constitutes a huge success, it is not a perfect algorithm. Eberhart has ever pointed out that the original PSO could eventually converge to a local minima (i.e. premature problem) similar to other evolutionary algorithms. Many investigations have been undertaken in order to deal with this possible disadvantage. The question is: "Are these variants good and robust enough for most of the optimization problems, especially for real-life problems?" Based on the No Free Lunch Theory. There will always be ways of amending its performance in certain fields of optimization. It shall be noted that now more and more concepts relied on in the field of PSO so as to compensate for its natural shortcomings, to increase its robustness and to improve its performance. In this spirit, two new concepts are introduced in this chapter that were developed during the course of this work .

Because of the complexity of the mathematical issues in stochastic algorithms, there is no coherent theoretical convergence proof of PSOs to date. Those proofs normally use a reduced form of PSO or get a convergence conclusion under the condition that the PSO could find the global optimum if it could run without any limit on the number of iterations. However, in most cases, we cannot afford to wait for such long time. As it does not seem to be possible to prove that an algorithm is able to perform sufficiently well over a wide range of feasible functions,the most common strategy to evaluate the performance of PSOs is to use a benchmark test comprising several functions. Although these functions may not necessarily supply an accurate description of the performance of an algorithm on real-world problems, they can be used to investigate certain aspects of the algorithms under consideration. Thus, these two modified PSOs are assessed using a standard benchmark test containing all characteristics which are difficult to evolutionary algorithms.

3.1 MGCPSO

Let us think about the PSO in another way. After several iterations, the particle's current individual best position will converge to the global best position, so that the second and third parts of formula (2.18) tend to be zero. The update process only depends on the momentum part, which may potentially constitute a risk for the PSO algorithm. If the global best position is close to a local minimum, PSO will converge to that point. Even more, it can

not be guaranteed that the result is a local minimum - it merely means that all the particles converge to a global best position (i.e. equilibrium point) discovered so far by the whole swarm, which is in line with the conclusion from section 2.4. Many variants have been discussed in section 2.5 that deal with this drawback in differing ways. They all tend to view the maintaining of a diverse swarm as the solution though. Since none of these variants can be recognized as a perfect algorithm, the performance of most of them can usually be improved by changing the parameter set, introducing new parameters or hybridizing two or more of them. In this section, a new variant called MGCPSO is introduced which is inspired by Guaranteed Convergence PSO (GCPSO) that was proposed by Van Den Bergh [35].

As was discussed in subsection 2.5.2.1, the basic idea of GCPSO is to introduce an additional particle τ which searches the region around the current global best position, i.e. its local best position is equal to the current global best position. The diameter $\rho(t)$ of the search area is dynamically adapted based on the behaviour of the swarm, i.e. if the swarm always finds a better position than the current global best position in consecutive iterations, the search diameter $\rho(t)$ will become larger; if the swarm always fails to find a better position than the current global best position in consecutive iterations, the search diameter $\rho(t)$ will become smaller. This approach is a good supplement to the original PSO and it is important to note that for the GCPSO algorithm all particles except for the global best particle still follow formula (2.18) and (2.16). Only the global best particle follows the new velocity and position update equations. According to [35] and Peer et al. [2003], GCPSO generally performs better (e.g. faster convergence) than standard PSO when applied to benchmark problems. This improvement in performance is particularly noticeable when applied to uni-modal functions, although the performance of both algorithms is generally comparable for multi-modal functions [35]. Furthermore, due to its fast rate of convergence, GCPSO is slightly more likely to be trapped in local optima [35]. However, it has guaranteed local convergence whereas the original PSO does not. This could be explained as if the current global best position is located on a local minimum which is not near the global minimum. The additional particle could possibly fail to find any better position around this local minimum and the algorithm would converge to the local minimum eventually.

The new approach MGCPSO is proposed to avoid this flaw and promote the performance of the original PSO. This approach reverses the GCPSO, i.e. if the swarm always finds a better position than the current global best position in consecutive iterations, the search diameter $\rho(t)$ will become smaller; if the swarm cannot find a better position than the current global best position in consecutive iterations, the search diameter $\rho(t)$ will become larger. This improvement can be explained as: When the consecutive number of successful search is increasing, it means that the swarm is probably searching closing to a minimum, concentrating the searching diameter will help the swarm to discover this position faster; If the swarm fails to find better positions in following iterations, in this case, the swarm may already lock a minimum and mark it as **b**, increasing the searching diameter of the additional particle can help the swarm to jump out of the trap of the local minimum and to search in a wide area. The validity of this approach is put to the test in a prominent analytical benchmark test in order to prove its performance which is discussed in section 3.6.

The update formula for particle τ is the same with formula (2.30), which is expressed as:

$$\mathbf{v}^{\tau}(t+1) = -\mathbf{x}^{\tau}(t) + \mathbf{b}(t) + \omega \mathbf{v}^{\tau}(t) + \rho(t)(1-2\mathbf{r}) \tag{3.1}$$

$$\mathbf{x}^{\tau}(t+1) = \mathbf{v}^{\tau}(t+1) + \mathbf{x}^{\tau}(t) \tag{3.2}$$

The new update formula for the diameter $\rho(t)$ is

$$\rho(t+1) = \begin{cases} 0.5\rho(t) & \text{if \#successes} > s_c \\ 2\rho(t) & \text{if \#failures} > f_c \\ \rho(t) & \text{otherwise} \end{cases} \tag{3.3}$$

The threshold parameters are $s_c = 15$, $f_c = 5$. The reason to choose a large value of s_c is that in high-dimensional problems, it is normally difficult to obtain better values using random search in only a few iterations, in particular around a local minimum, so that the algorithm is quicker to punish a poor set of ρ than it is in rewarding a successful ρ value. When ρ becomes sufficiently small, the additional particle will stop searching which is not what we expect. On the other hand, when ρ is too large, the particle τ will tend to fly away from the design area. Thus the best choice for this algorithm is to treat ρ in the same manner as the particle's velocity, i.e. a limit for ρ will be set empirically as $\rho(t) \in (\rho_{min}, \rho_{max}$, which together with the velocity limit will constrain the updated velocity in a reasonable interval. The following modification could be made to this approach: For multi-modal or non-convex problems, the number of the additional particles τ can be increased in order to enlarge the searching ability of the swarm.

Moreover, a mechanism inspired by Guaranteed Global Convergence PSO (SPSO) is introduced to maintain the diversity of the swarm. If particle i's position $\mathbf{x}^i$ is to close to the global best position so far discovered $\mathbf{b}$, then it will be randomly regenerated. Thus, the whole work flow of the MGCPSO can be seen below:

1. Initialize
   (a) Set the optimal parameters: s_c, f_c, $\rho(0)$ and random seed
   (b) Generate a swarm with s particles randomly distributed in the design domain
   (c) Generate the initial velocities randomly for each particle, $0 \leq \mathbf{v}_j^i(0) \leq v_{max}$
   (d) Evaluate fitness values for each initial particle $f(\mathbf{x}^i(0))$ and set $\mathbf{p}^i(0) = \mathbf{x}^i(0))$
   (e) Find the global best position $\mathbf{b}(0) = \{\mathbf{x}(0) \mid \min f(\mathbf{x}^i(0)), \quad i \in \mathrm{S}\}$
2. Optimization
   (a) For each particle $i \in \mathrm{S}$

   If $\| \mathbf{x}^i(t) - \mathbf{b}(t) \| \leq \epsilon$ where ϵ is a user defined parameter, then
   Randomly generate $\mathbf{x}^i(t+1)$
   Else
   Update particle's velocity $\mathbf{v}^i(t+1)$ using formula (2.18)
   Update particle's position $\mathbf{x}^i(t+1)$ using formula (2.16)
   End if

(b) Update $\rho(t+1)$ using formula (3.3).

(c) Update $\mathbf{x}^{\tau}(t+1)$ using formula (3.1) and (3.2).

(d) For each particle $i \in S$

Evaluate fitness value using coordinates $\mathbf{x}^i(t+1)$ in design space
If $f(\mathbf{x}^i(t+1)) \leq f(\mathbf{p}^i(t))$, then
Set $\mathbf{p}^i(t+1) = \mathbf{x}^i(t+1)$
Else
Set $\mathbf{p}^i(t+1) = \mathbf{p}^i(t)$
End if
If $f(\mathbf{p}^i(t+1)) \leq f(\mathbf{b}(t))$, then
Set $\mathbf{b}(t+1) = \mathbf{p}^i(t+1)$
End if

(e) If stopping criteria is satisfied, go to 3; other wise go to 2

3. Terminate, export result

3.2 LPSO

As we discussed in subsection 2.3.5, particle swarm algorithms have historically been studied in two general types of neighborhoods, called *Gbest* and *Lbest*, which are illustrated in figure 2.3. The initial study on topologies of the PSO was conducted by Kennedy in order to obtain information on their influence on the performance of PSO. Three topologies, including ring topology, wheel topology and *Gbest* topology were studied.

In ring topology, the distant parts of particles are also independent of one another. Neighbours, on the other hand, are closely connected. Thus, one segment of the population might converge to a local minimum, while another segment converges to a different optimum or keeps seeking the optimum. Information is spread from neighbour to neighbour in this topology, until the true optimum is found.

In wheel topology, on the other hand, individuals are effectively isolated from one another as all information has to be communicated through the focal individual. This focal individual compares performances of all the particles and moves towards the very best of them. If this moving direction could promote the performance of the focal individual, then that performance is eventually communicated to the rest of the swarm. Thus, the focal individual can be described as a buffer or filter that lowers the rate of transmission of important information in the population. Whilst it is a way of maintaining the diversity of potential solutions to the problem, it may also entirely destroy the ability of the swarm to collaborate.

It shall be noted that this study only used standard PSO. The main conclusions of this study are that *Gbest* seems faster but that it is more vulnerable to local optima whereas *Lbest* appears much slower but more robust in the face of an increased number of iterations. Wheels performed badly except on one of the functions. However, results cannot be deemed comprehensive.

After that Kennedy and Mendes proposed a new PSO model using a Von Neumann topology. Research shows that PSO using Von Neumann topology will slow down the convergence rate as well as using the ring topology because the best solution found has to propagate through several neighborhoods before affecting all particles in the swarm. This slow propagation will enable the particles to explore more areas in the search space and thus decreases the chance of premature convergence even with a small amount of particles.

Because the information links are changed in PSOs based on the *lbest* topologies, the velocity updating formula has to be changed correspondingly. In the *lbest* model, the whole swarm can be thought of as being divided into several sub-swarms and each individual communicates only with its neighbors which is defined by topologies of the information links. Thus the global best position $\mathbf{b}(t)$ in formula (2.18) so far discovered in *Gbest* model has to be changed to local best position $\mathbf{b}^i(t)$ so far discovered by sub-swarm $\mathbb{S}_j$ in *lbest* model correspondingly. The velocity updating formula for *lbest* model is now expressed as:

$$\mathbf{v}^i(t+1) = \omega\mathbf{v}^i(t) + C_1R_1(\mathbf{p}^i(t) - \mathbf{x}^i(t)) + C_2R_2(\mathbf{b}^i(t) - \mathbf{x}^i(t)) \tag{3.4}$$

It is important to realize that most research work done on the improvement of PSOs that are based on *lbest* topologies has been restricted to to the designing of different topologies of the population. However these variants cannot obtain good solutions when applied to highly multi-modal problems, such as truss topological optimizations. In this section, a modified PSO based on *Lbest* topology is proposed by adding a new rule to the position updating procedure, which is inspired by the **Guaranteed Global Convergence Particle Swarm Optimizer** (SPSO).

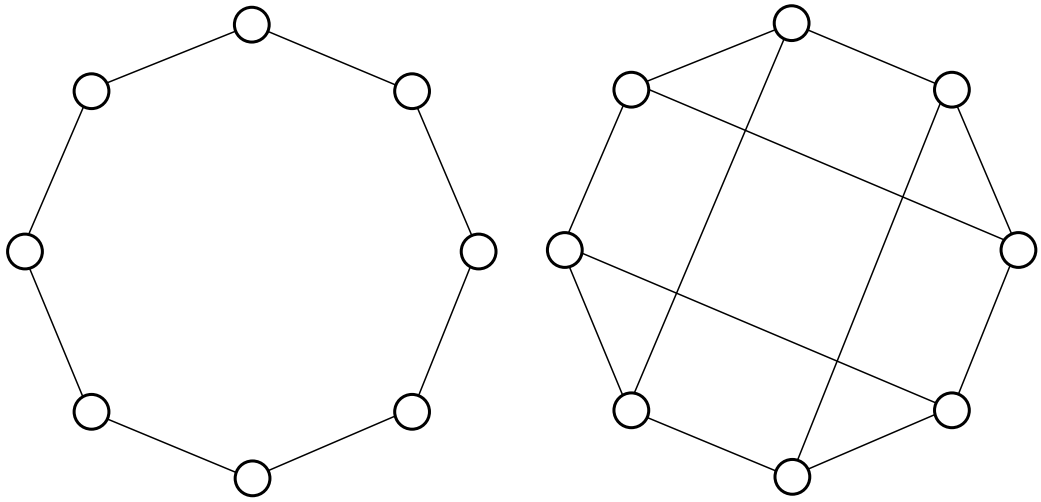
Note that in formula (3.4), if for particle i on time step t, $\mathbf{x}^i(t) = \mathbf{p}^i(t) = \mathbf{b}^i(t)$, its new updated velocity will be $\mathbf{v}^i(t+1) = \omega\mathbf{v}^i(t)$, it means that particle i will move following its previous track, especially during the later evolution iterations. Most of the particles cluster around this global best position and their velocities are relatively small compared with their initial ones so that eventually all the particles will converge to this point, even though it may be not an optimum which would reduce the particle's searching ability. This disadvantage is the main reason for the problem of prematurity that attaches to PSO. For *lbest* PSO, each particle has its own local best position $\mathbf{b}^i$. In order to set a convenient stopping criterion, a variable $\mathbf{b}(t)$ from *Gbest* based PSO is included, called current global best position, which is defined as:

$$f(\mathbf{b}(t)) \leq f(\mathbf{b}^i(t)), \quad \forall i \in \{1, s\} \tag{3.5}$$

Now, the stopping criterion can be expressed as: if $\mathbf{b}(t)$ are not being updated in n consecutive iterations, the program will stop running, which is the same with the stopping criterion of *Gbest* based PSO.

In this new approach, in order to improve the searching ability of *Lbest* based PSO, two new mechanisms are added to a particle's evolution procedure:

1. in case that the condition $\|\mathbf{x}^i(t) - \mathbf{b}^i(t)\| < \epsilon$ is satisfied in continuous n iterations, where ϵ is a predetermined small value to determine if $\mathbf{x}^i(t)$ is much close to $\mathbf{b}^i(t)$ and n is an integer to determine if a particle could find a better solution in a very



(a) Ring Topology with $K = 2$ (b) Ring Topology with $K = 3$

Figure 3.1: two Ring Topologies

small region around $\mathbf{b}^i(t)$, the particle i's position for next iteration $\mathbf{x}^i(t+1)$ will be randomly generated.

2. further more, if $f(\mathbf{x}^i(t)) < f(\mathbf{b}^i(t-1))$, $\mathbf{b}^i(t)$ is updated to $\mathbf{x}^i(t)$ and the particle i's best individual position ($\mathbf{p}^i(t)$) is not replaced by $\mathbf{x}^i(t)$.

For other particles which do not match these conditions are manipulated according to formula (2.18). It is noted that these two mechanisms are used to maintain the diversity of the swarm and improve the particle's searching abilities. The purpose of the first one is to avoid the particle's accumulating phenomenon in later phases of the evolution procedure. The second one can avoid $\mathbf{p}^i$ and $\mathbf{b}^i$ colliding each other, thus directions of the "memory" part and the "cognitive" part in particle's velocity update formula (3.4) keep different, which can assure that the particles' trajectories are always affected by three different directional vectors if their positions are updated via formula (2.16).

The ring topology is used for the proposed variant due to its superior performance compared with other *Lbest* topologies. In ring topology, each individual interacts with their k nearest neighbors (k can be selected from $\{2, \cdots, s-1\}$, where s is the total amount of particles. If $k = s$, *lbest* topology is automatically transformed into *Gbest* topology). In this work, *Lbest* topologies with $k = 2$ and $k = 3$ are studied for this variant. These topologies are shown in figure 3.1. The whole work flow of the LPSO is seen below:

1. Initialize
   (a) Set ϵ, stopping condition n, random seed and create ring topology
   (b) Generate a swarm with s particles randomly distributed in the design domain
   (c) Generate the initial velocities randomly for each particle, $0 \leq v_j^i(0) \leq v_max$
   (d) Evaluate fitness values for each initial particle $f(\mathbf{x}^i(0))$ and set $\mathbf{p}^i(0) = \mathbf{x}^i(0)$

(e) Find the best local best position $\mathbf{b}^i(0) = \{\hat{\mathbf{x}}(0) \mid \min f(\mathbf{x}^j(0)), \quad j \in \mathbb{S}_i\}$

(f) Find the current global best position $\mathbf{b}(0) = \{\hat{\mathbf{p}}(0) \mid \min f(\mathbf{p}^j(0)), \quad j \in \{1, \cdots, s\}\}$

2. Optimization

(a) For each particle $i \in \mathbb{S}$

Evaluate fitness value using coordinates $\mathbf{x}^i(t+1)$ in design space

If $\parallel \mathbf{x}^i(t) - \mathbf{b}^i(t) \parallel \leq \epsilon$ and $\mathbf{b}^i$ can not be updated in n continuous iterations then

Randomly generate $\mathbf{x}^i(t+1)$

Else

Update particle's velocity $\mathbf{v}^i(t+1)$ using formula (3.4)

Update particle's position $\mathbf{x}^i(t+1)$ using formula (2.16)

Endif

(b) For each particle $i \in \mathbb{S}$

If $\mathbf{x}^i(t+1) < \mathbf{b}^i(t) < \mathbf{p}^i(t)$ then

Set $\mathbf{b}^i(t+1) = \mathbf{x}^i(t+1)$

Set $\mathbf{p}^i(t+1) = \mathbf{p}^i(t)$

Else if $\mathbf{b}^i(t) \leq \mathbf{x}^i(t+1) < \mathbf{p}^i(t)$ then

Set $\mathbf{b}^i(t+1) = \mathbf{b}^i(t)$

Set $\mathbf{p}^i(t+1) = \mathbf{x}^i(t+1)$

Else

Set $\mathbf{b}^i(t+1) = \mathbf{b}^i(t)$

Set $\mathbf{p}^i(t+1) = \mathbf{p}^i(t)$

End if

(c) Update $\mathbf{b}(t+1) = \{\hat{\mathbf{p}}(t+1) \mid \min f(\mathbf{p}^j(t+1)), \quad j \in \{1, \cdots, s\}\}$

(d) If stopping criteria is satisfied, go to 3; other wise go to 2

3. Terminate, export result

3.3 Parallelizing modified PSOs

Because of its belonging to the family of evolutionary algorithms, its greatest drawback is the great cost of computing over time. One approach to reduce the elapsed time is to make use of coarse-grained parallelization to evaluate the design points. In this section, a synchronous parallel pattern for the proposed modified PSOs will be described.

3.3.1 Parallel Computing

The introduction of fast computers has given rise to a new way of doing scientific work. The two classic branches of theoretical and experimental science are complemented by computer science. Previous successes in the realm of computer science over the past decade have created an everlasting demand for the development of the supercomputer.

On one hand, computer scientists have achieved great advances with regard to microprocessor technology over the past ten years. To illustrate this, Clock rates of processors have increased from about 40 MHz to over 3.0 GHz. At the same time, processors are now able to execute multiple instructions in one and the same cycle. The average number of cycles per instruction (CPI) of high end processors has improved by roughly an order of magnitude over the past 10 years. The increase of clock rates still continues along the lines of Moore's Law which can be stated in the following terms [56]:

> The number of transistors that can be placed inexpensively on an integrated circuit has increased exponentially, doubling approximately every two years.

However the microprocessor technology is restricted by the following limits (both physical and practical):

1. **Transmission speeds**: The speed of a serial computer is directly dependent on how fast data can move through hardware. Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increased speed necessitates an increased proximity of processing elements.

2. **Limits to miniaturization**: Processor technology is allowing an increased amount of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be in the future.

3. **Economic limitations**: The cost of advanced single-processor computers increased more rapidly than their power. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

These limitations show that the speed of CPU cannot be increased indefinitely and Moore's Law will expire some day in the future. This seems very gloomy to the future of computer science. Whenas an evolution called "parallel computing" has been issued during this time from experimental conceptions in laboratories to the everyday tools of computational scientists who always need the ultimate in computer resources in order to solve their problems. The term "parallel computing" refers to a form of computation in which many calculations are carried out simultaneously, based on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel"). In practice, this can be accomplished by dividing the problem into ideally small parts so that each processing element can execute its part of the algorithm simultaneously with the others. The processing elements can be diverse and may contain resources such as a single computer with multiple

processors, several computers connected with networks, specialized hardware, or any combination of the above. In the simplest sense, parallel computing is the simultaneous use of multiple computer resources to solve a computational problem.

In this work, message passing model is chosen to parallelize the modified PSOs from all above model, while message passing model has the following advantages compared with other parallelism approaches:

1. **University**: The message-passing model fits well on separate processors connected by a communication network. Thus it matches the hardware of most of the existing parallel supercomputers, as well as the workstation networks and dedicated PC clusters, i.e. this model is portable and scalable and the number of computation nodes does not pose a limit.

2. **Expressivity**: The message-passing model is a useful and complete model to express parallel algorithms intuitively.

3. **Ease of debugging**: A difficult area of parallel computing is debugging. The message-passing model, by controlling memory references more explicitly than any of the other models (only one process has direct access to any memory location), makes it easier to locate erroneous memory reads and writes. Moreover, some debuggers can show the message queues which are normally invisible to the programmers

4. **Performance**: Performance is the most important reason why the message passing model will remain a permanent part of the parallel computing environment. It provides a way for the programmers to explicitly associate specific data with processes and thus allow the complier and cache-management hardware to function fully.

3.3.2 Parallelizing PSOs

Present day engineering optimization problems often pose large computational demands, resulting in long computation times even when using a modern high-end processor. In order to obtain enhanced computational throughput and global search capability, it is necessary to design a parallel pattern for the particle swarm optimization algorithm which is an increasingly popular global search method. In this subsection, two parallelizing programs are offered with respect to the two proposed PSOs discussed in section 3.1 and 3.2.

The concern of the greatest importance in parallel programming is the decomposition of the computing task into a set of tasks for concurrent execution. For PSOs, as we have discussed, the most time-consuming procedure is the evaluation of fitness functions especially for a complex large-scale optimization problem. Furthermore, this part of the computational work is scalable, i.e. increasing the number of particles will cause the amount of evaluations to increase proportionally. Thus, this part of the computation load can be partitioned based on a particle decomposition method. Compared with the other computation load partitioning method, for example the space decomposition method which divides the load based on the problem space, the particle decomposition method can reduce the amount of information to be exchanged among processors.

The program can be parallelized through a master-slave model. In this model, there is only one master node which performs the same computation tasks as other slave nodes and additionally serves as an information analyzing and exchange agent. Each slave nodes is responsible for evaluating the fitness function relating to the corresponding particles, for updating the information concerning their best individual positions. Besides, the master node needs to collect the information about particle's individual best position sent to it by the slave nodes, update the best global position and scatter this information to slave nodes. The parallel model in this work is inspired by the synchronous parallel algorithm proposed by Schutte et el. [118]. In the parallel model from J. F. Schutte, the master node stores all the information of all particles, including particles' positions, velocities, individual best positions and global best positions, it determines not only the global best position $\mathbf{b}(t)$ for the whole swarm but also the individual best position $\mathbf{p}(t)$ for each particle, Thus the task of determining the individual best position $\mathbf{p}(t)$ and the particle's new positions $\mathbf{x}(t)$ can be time-consuming, especially when lots of particles are involved. However, in our approach, an enhancement is proposed meaning that these two operations are also decomposed into each process. Data transferring between master nodes and slave nodes is now the individual best position with corresponding fitness value and the global best position instead of particle's position with corresponding fitness value. It shall be noted that in our approach a sequential random generator is used to produce all of particles' initial positions. In order to acquire correct normal distributed positions, this manipulation is done only by the master node in the first iteration and the particles' date is then scattered into slave nodes. The flow chart of the parallel PSO is shown in figure 3.2.

Parallelizing MGCPSO Let us recall the workflow of MGCPSO, the newly included feature is to add a particle τ which could be treated as a normal particle. Thus we can use the same paradigm to parallelize the MGCPSO with just a small change: a different updating formula, as well as additional corresponding parameters should be stored in the same node that particle τ is kept in. The flow chart of the parallelizing PSO is shown in figure 3.3.

Parallelizing LPSO This modification of PSO differs from *Gbest* based PSOs in that each particle does not use a uniform $\mathbf{b}(t)$ but local $\mathbf{b}^i(t)$ to update its velocity and $\mathbf{b}^i(t)$ is dependent on the topology of the information link between particles. This is shown in figure 2.3. Since in LPSO, $\mathbf{b}^i t$ is determined priori to $\mathbf{p}^i$ due to the second new mechanism discussed in subsection 3.2, the sequence of updating these two dates has to be reversed. Thus in the parallel paradigm for LPSO, the master node first determines both the global best position $\mathbf{b}(t)$ and the local best position $\mathbf{b}^i(t)$ based on the topology of the information link and sends them to corresponding slave nodes. In case some particle needs to be regenerated according to the new inserted rules, the master node will also execute this manipulation and send it to the corresponding slave node. After, each slave node updates the individual best positions $\mathbf{p}^i t$. The flow chart of the parallelizing PSO is shown in figure 3.4.

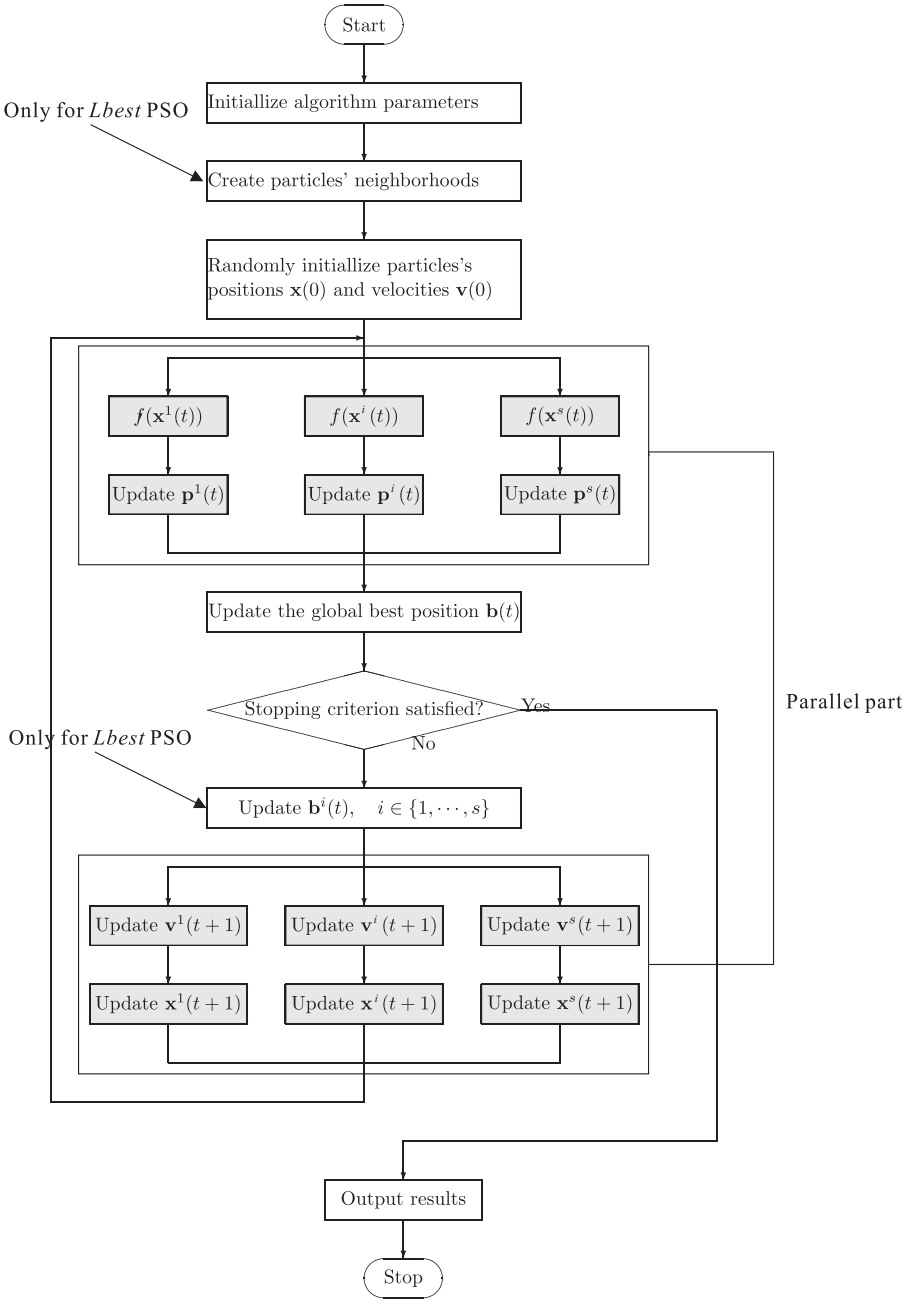


Figure 3.2: Parallel paradigm of canonical PSO

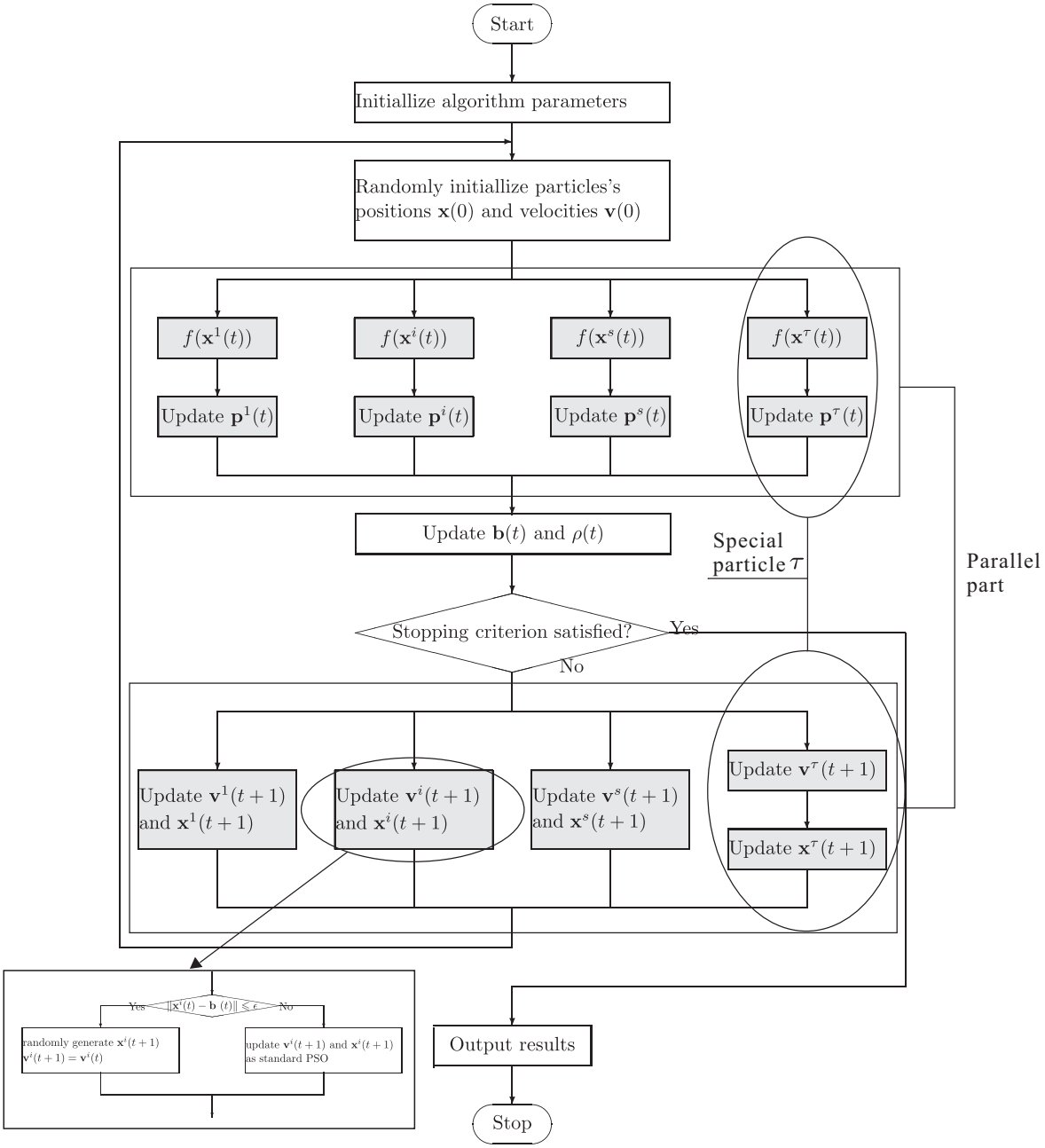


Figure 3.3: Parallel paradigm of MGCPSO

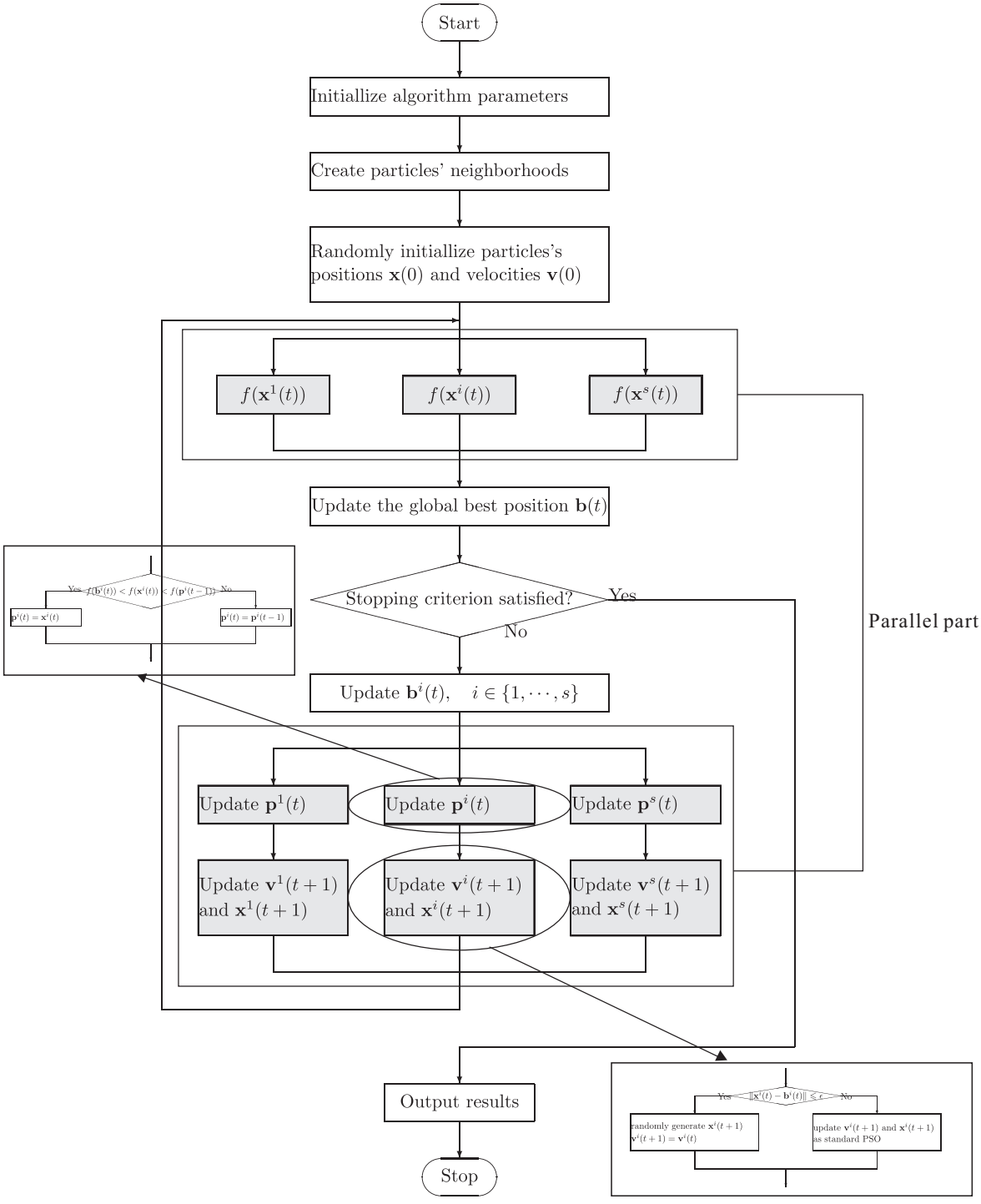


Figure 3.4: Parallel paradigm of LPSO

3.4 Description of Benchmark suite

This is a very popular suite comprising test functions which are commonly used to test the performance of evolutionary algorithms. These functions are chosen with regard to their particularities, which include several scenarios, from one simple function with single minimum to one having a considerable number of local minima with similar fitness values. Each function is illustrated in two figures (one for the full search space, another one for the area around the global minimum) for two-dimensional cases except the quadratic function.

3.4.1 Quadratic

This is a simple case with only one minimum. Any optimization technique should solve it without problem, however, PSOs may not be as effective as a specific deterministic algorithms (e.g. gradient based optimization algorithms) due to their stochastic properties. Its simplicity helps to focus on the effects of dimensionality in optimization algorithms. Its global minimum locates at $\mathbf{x} = [0, \cdots, 0]^n$ with $f(\mathbf{x}) = 0$.

Objective Function:	$f(\mathbf{x}) = \sum_{i=1}^{n} \mathbf{x}_i^2$
Search Space:	$\{x_i \mid \forall i : -100 < x_i < 100\}$
Dimensionality:	30
Global Minimum:	at $x_i = 0$

3.4.2 Rosenbrock

In mathematical optimization, the Rosenbrock function is a non-convex function used to test the performance of optimization algorithms. It is also known as Rosenbrock's valley or Rosenbrock's banana function. The global minimum is inside a long, narrow, parabolic shaped flat valley. Its variables are strongly dependent and gradient information often misleads algorithms. As a results, converging to the global is difficult. It is noted that if $n > 3$, there is at least one local minimum close to $\mathbf{x} = [-1, 1, \cdots, 1]^n$ in addition to the global minimum $\mathbf{x} = [1, 1, \cdots, 1]^n$ with $f(\mathbf{x}) = 0$.

Objective Function:	$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[(1 - x_i)^2 + 100(x_i^2 - x_{i+1})^2\right]$
Search Space:	$\{x_i \mid \forall i : -10 < x_i < 10\}$
Dimensionality:	30
Global Minimum:	at $x_i = 1$

3.4.3 Ackley

Originally, this problem was initially defined for two dimensions [6] which is apparently like a little pine, but the problem has been generalized to N dimensions [11]. It is a difficult problem, even with the small dimensionality. The basin of attraction of the global minimum is very narrow, so that optimization algorithms can easily be trapped in a local minimum

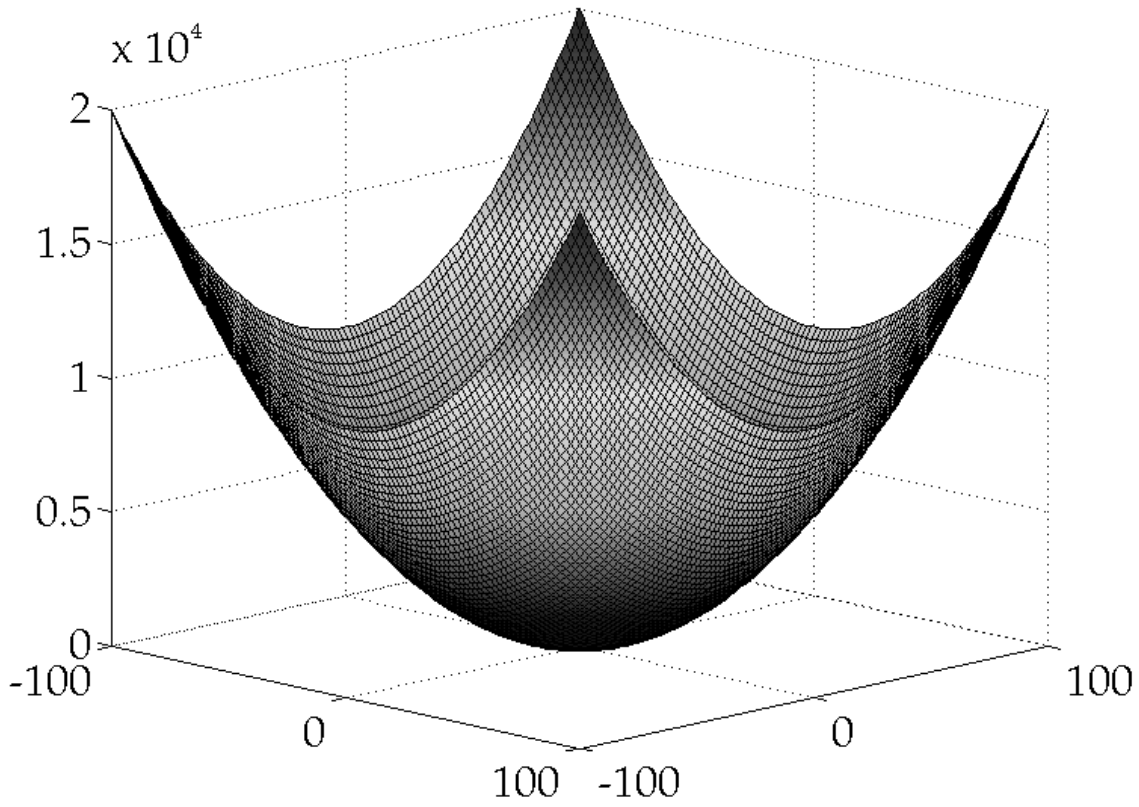


Figure 3.5: Graph of the Quadratic function in two dimensions

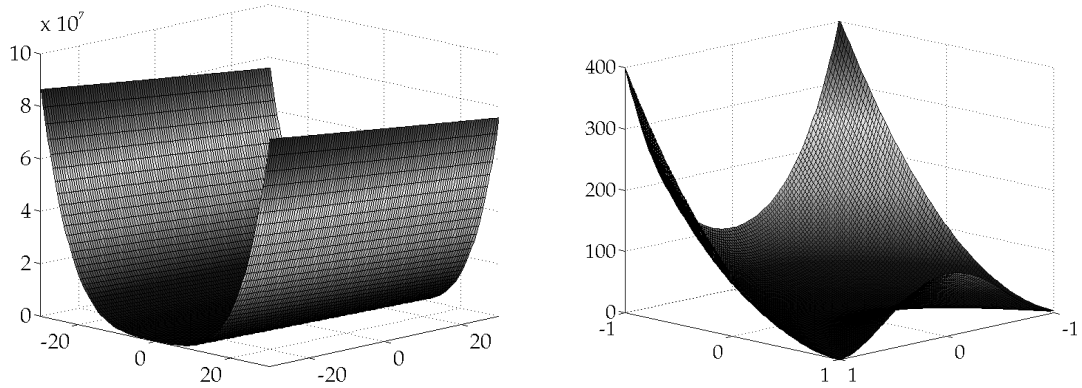


Figure 3.6: Graph of the Rosenbrock function in two dimensions

on their way to the global minimum. Its global minimum locates at $\mathbf{x} = [0, \cdots, 0]^n$ with $f(\mathbf{x}) = 0$

Objective Function:	$f(\mathbf{x}) = -20e^{-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}} - e^{\frac{\sum_{i=1}^{n} \cos(2\pi x_i)}{n}} + 20 + e$
Search Space:	$\{x_i \mid \forall i : -30 < x_i < 30\}$
Dimensionality:	30
Global Minimum:	at $x_i = 0$

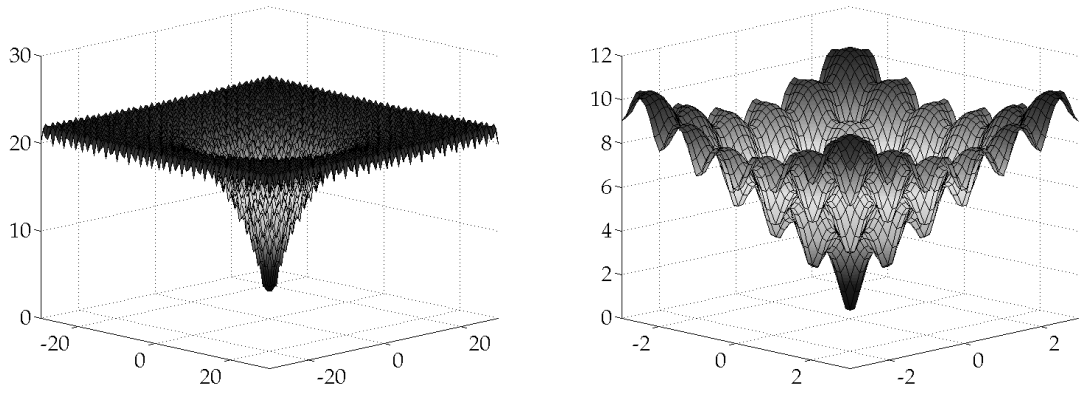


Figure 3.7: Graph of the Ackley function in two dimensions

3.4.4 Rastrigin

The Generalized Rastrigin Function is a typical example of a non-linear multimodal function, which is characterized by deep local minima arranged as sinusoidal bumps. The global minimum is $\mathbf{x} = [0, \cdots, 0]^n$ with $f(\mathbf{x}) = 0$. This function is a fairly difficult problem due to its large search space and its large number of local minima. An optimization algorithm can easily be trapped into a local minimum.

Objective Function:	$f(\mathbf{x}) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos 2\pi x_i + 10\right]$
Search Space:	$\{x_i \mid \forall i : -5.12 < x_i < 5.12\}$
Dimensionality:	30
Global Minimum:	at $x_i = 0$

3.4.5 Griewank

This function is strongly multi-modal with significant interaction between its variables caused by the product term. The global optimum is $\boldsymbol{x} = [100, \cdots, 100]^n$ with $f(\mathbf{x}) = 0$, which is almost indistinguishable from closely packed local minimum that surround it. This function has the interesting property that the number of local minima increases with dimensionality. On the one hand, this tends to increase the difficulty of the problem but on the other hand, because the distances between two adjacent local minimum are quite small, it is very easy to escape from them, in the case of stochastic algorithms.

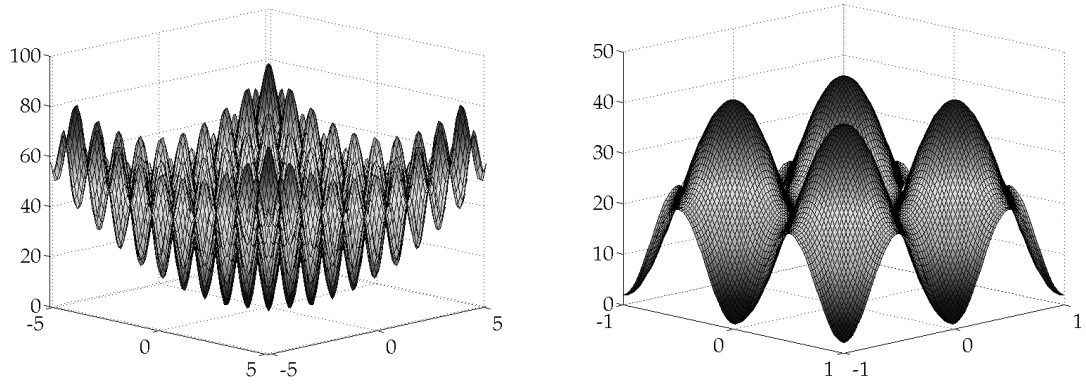


Figure 3.8: Graph of the Rastrigin function in two dimensions

Objective Function:	$f(\boldsymbol{x}) = 1 + \frac{\sum_{i=1}^{n}(\boldsymbol{x}_i - 100)^2}{4000} - \prod_{i=1}^{n} \cos(\frac{\boldsymbol{x}_i - 100}{\sqrt{i}})$
Search Space:	$\{\boldsymbol{x}_i \mid \forall i : -300 < \boldsymbol{x}_i < 300\}$
Dimensionality:	30
Global Minimum:	at $x_i = 0$

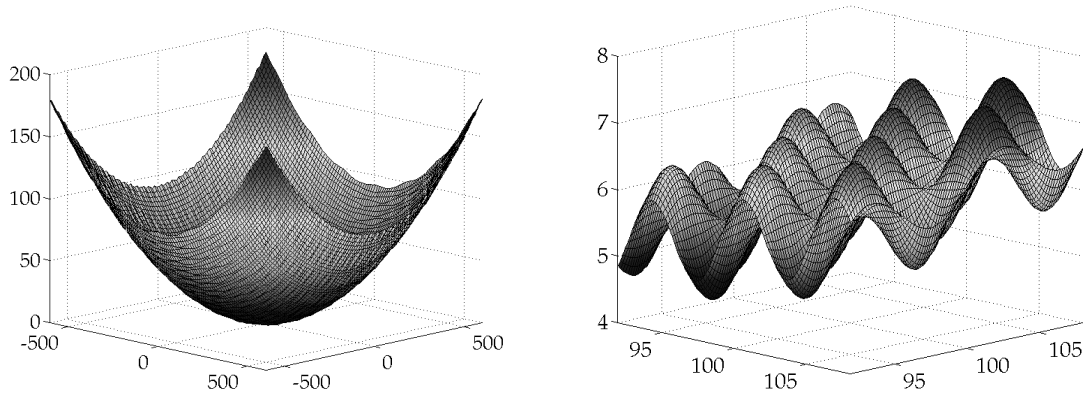


Figure 3.9: Graph of the Griewank function in two dimensions

3.4.6 Schaffer F6

This is a very difficult two-dimensional function, optimization algorithms would like to trap into its local minima which is arranged in concentric circles around the global optimum which itself is located in a narrow basin. The global optimum is $\mathbf{x} = [0, 0]$ with $f(\mathbf{x}) = 0$.

Objective Function:	$f(\mathbf{x}) = 0.5 + \frac{\sin(\sqrt{{x_1}^2 + {x_2}^2})^2 - 0.5}{(1 + 0.001({x_1}^2 + {x_2}^2))^2}$
Search Space:	$\{x_i \mid \forall i : -100 < x_i < 100\}$
Dimensionality:	2
Global Minimum:	at $x_i = 0$

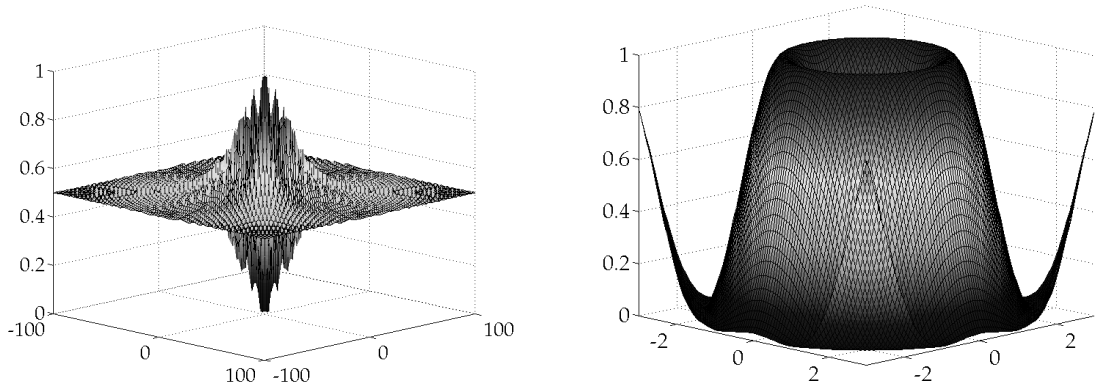


Figure 3.10: Graph of the Schaffer function in two dimensions

3.5 Methodology

Stochastic algorithms, including evolutionary methods, some gradient decent algorithms and simulated annealing can perform differently each time they are run. If a landscape has a number of hills, algorithms start in a good or bad region and proceed from there to a better or worse place in the search space. Thus in order to evaluate the performance of optimization algorithms in a given problem, it is necessary to collect enough data to carry out a statistical analysis with a certain degree of accuracy. It is important to have enough statistical power to be able to perform any kind of statistical conclusion.

3.5.1 Statistic Values for measurement

According to the central limit theorem, the mean of a given population can only be approximated by the normal distribution when the sample size is bigger than 30. Thus, initial particles of the swarm are distributed throughout the search space using a uniform random number generator, so that the value of each dimension j of particle i can be sampled as

$$x_j^i \sim U(-d, d) \tag{3.6}$$

where d is the appropriate boundary value for the specific dimension of search space in terms of the function under consideration. Current practice in EC and several fields of Machine Learning neglects crucial aspects when presenting statistical results of the experiments [50]. Therefore, the following measurements are used to evaluate the performance of PSOs:

1. Mean Value f_m: It is the arithmetic average of all the results from the tests on the same problem, which is denoted as:

$$f_m = \frac{1}{n}\sum_{i=1}^{n} f_i \tag{3.7}$$

where n is the total amount of numerical tests and f_i is the result for the i_th test. This value describes the average performance of an algorithm intuitively.

2. Best Result f_b: It is the best value obtained from all of the tests, which is a very important measurement for global optimization algorithms and used to assess the search ability of an algorithm, i.e. whether this algorithm can find a global optimum or a better position compared with the best known result or not.

3. Worst Result f_w: It is the worst solution from all of the tests, which can be used to evaluate the stability of an algorithm.

4. Mean Number of Iterations i_m: It is the average iteration number of all the tests on the same problem, which describes the efficiency of an algorithm.

5. Speed-up Ratio S_p: It is only used for parallelizing PSOs and describes how much faster a parallel algorithm is faster than a corresponding sequential algorithm which is defined as:

$$S_p = \frac{T_s}{T_p} \tag{3.8}$$

where T_s and T_p refer to the execution time of the sequential algorithm and the execution time of the parallel algorithm with p processors respectively. Linear speedup or ideal speedup is obtained when $S_p = p$. Good Speed-up means good scalability of a parallel algorithm, however, linear speedup cannot be obtained for all of the parallel algorithms due to communication latency.

The following conclusions are based on the aforementioned statistical values.

3.5.2 Parameter Selection and Test Procedure

The parameters of PSOs used for benchmark tests are the following:

1. **Inertia Weight** ω: This coefficient is used to control the trajectories of particles and set $\omega = 0.5 + rand()$ where $rand()$ is a random number generator.

2. **Acceleration Coefficients** φ_1 and φ_2: These are mostly used in the community of particle swarms, the values are $\varphi_1 = \varphi_2 = 1.49445$.

3. **Maximum Velocity** v_{max}: This parameter was not set, as it was not deemed necessary.

4. **Population Size**: All the swarms used in this study comprise twenty individuals. There is no universal rule to determine the number of particles in a swarm, normally, it is equal to or less than the number of unknowns. In this benchmark test, twenty is only an empirical value.

5. **Stopping criterion**: If the best position of the swarm cannot be improved in fifty consecutive iterations the program will be stopped artificially and the fitness of the best position will be considered as the result of this numerical test.

		mean value	best value	worst value	Mean Number of Iterations
Sphere Function	PSO	0	0	0	1320
	MGCPSO	0	0	0	1376
	LPSO=2	0	0	0	2574
	LPSO=3	0	0	0	2227

Table 3.1: Results from Quadratic function

		mean value	best value	worst value	Mean Number of Iterations
Rosenbrock Function	PSO	38.2609	4.5984	312.1458	2148
	MGCPSO	0.7992	0.001983	29.7098	2582
	LPSO=2	0.03059	0	0.6118	4126
	LPSO=3	0.1587	0.0004625	1.6269	3583

Table 3.2: Results from Rosenbrock function

6. **Maximum number of iterations**: It is another termination criterion. This is the maximum number of iterations during which the program is run. The measure collected at this time is whether the criterion was reached, i.e., a solution of a given quality, or not. This value was set at 5,000 iterations.

Each test function was tested thirty times independently and results and the number of iterations were collected for further statistical analysis.

3.6 Results and Conclusion

3.6.1 Algorithms' performances on Quadratic Function

From table 3.1, it should be noted that all the PSOs obtain the same results (including mean value, best value and worst value) from the quadratic function, which is a unimodal function. The sequence of convergence speed is shown:
PSO < MGCPSO < LPSO=3 < LPSO=2

3.6.2 Algorithms' performances on Rosenbrock Function

From table 3.2, it should be noted that the following five test functions are multimodal functions and the standard PSO is very easy to trap into local minima. For the rosenbrock function, only the PSOs based on a *lbest* model can find the global optimum. However, they need much more fitness function evaluations. The LPSO with $k = 2$ is more dominant in best result and worst result than $k = 3$ but with more iterations. MGCPSO has a competitive performance and the performance of the standard PSO is the worst. The sequence of convergence speed can be shown:
PSO < MGCPSO < LPSO=3 < LPSO=2

		mean value	best value	worst value	Mean Number of Iterations
Ackley Function	PSO	0.3059	0	1.4911	2871
	MGCPSO	0.003729	0	0.9680	3214
	LPSO=2	1.3028D-05	0	0.001616	4520
	LPSO=3	0.0001371	0	0.01610	3974

Table 3.3: Results from Ackley function

		mean value	best value	worst value	Mean Number of Iterations
Rastrigin Function	PSO	15.0	1.3936	27.93	1259
	MGCPSO	0.7075	0.0008741	4.044	1893
	LPSO=2	0.005376	0	0.019700	3012
	LPSO=3	0.0218	0	1.061	2436

Table 3.4: Results from Rastrigin function

		mean value	best value	worst value	Mean Number of Iterations
Griewank Function	PSO	0.01619	0.0005438	0.5790	2743
	MGCPSO	0.0002561	0	0.01902	3285
	LPSO=2	9.9611D-06	0	0.007534	4765
	LPSO=3	4.3535D-05	0	0.003610	3981

Table 3.5: Results from Griewank function

3.6.3 Algorithms' performances on Ackley Function

From table 3.3, it is noted that all the PSOs are able to find the global optimum. Considering also the mean, best and worst results, the LPSO with $k = 2$ is the most dominant among these PSOs, but it still has the most iterations. Although the standard PSO are able to find the global optimum, its mean result is not competitive compared with other PSOs and neither is its worst result. The sequence of convergence speed can be shown:
PSO < MGCPSO < LPSO=3 < LPSO=2

3.6.4 Algorithms' performances on Rastrigin Function

Table 3.4 shows that once again that only the PSOs based on the *lbest* model are able to detect the global optimum. Same to the results for the rosenbrock function, the LPSO with $k = 2$ is the most dominant among these PSOs. But it still has the problem of efficiency. The standard PSO is once again the last place if the mean iteration number is not involved. The sequence of convergence speed can be shown:
PSO < MGCPSO < LPSO=3 < LPSO=2

		mean value	best value	worst value	Mean Number of Iterations
Schaffer f6 Function	PSO	0.4650	0.009541	0.7942	520
	MGCPSO	0.01329	0	0.08023	891
	LPSO=2	2.8893D-05	0	0.004534	1530
	LPSO=3	0.008234	0	0.009794	1103

Table 3.6: Results from Schaffer f6 function

3.6.5 Algorithms' performances on Griewank Function

From table 3.5, it is noted that all the PSOs except standard PSO could are able to find the global optimum. If mean, best and worst results are considered, the LPSO with $k = 2$ is still the most dominant optimization approach among these PSOs. The MGCPSO has a positively balanced performance in relation to other PSOs And the sequence of convergence speed is shown as:
PSO < MGCPSO < LPSO=3 < LPSO=2

3.6.6 Algorithms' performances on Schaffer f6 Function

Table 3.6 shows that all the PSOs except standard PSO are able to find the global optimum. If mean result, best result and worst result are taken into consideration, the LPSO with $k = 2$ is once again the most dominant optimization approach among these PSOs. But it is not the most robust one due to its extreme number of function evaluations. The sequence of convergence speed can be shown:
PSO < MGCPSO < LPSO=3 < LPSO=2

3.6.7 Speed-up rates

In order to evaluate the performance of the parallel PSOs, 1, 2, 5, 10 and 20 computing processors are used because the swarm size is 20. For simplicity, the average speed-up rate of all the test functions was used to present the performance of the parallel implementation in the analytical test because it doesn't vary widely between different test functions. The speed-up rate of the analytical tests is almost linear.

3.6.8 Conclusion

From the above results, we see that the proposed MGCPSO and LPSO are two good supplements to the original PSO and both of them obtain competitive results compared with the standard PSO. The following generic conclusions can be drawn from these results:

1. Compared with other PSOs, the MGCPSO has a relatively well balanced performance, if all of the statistic values form the measurement except speed-up rate are taken into consideration. The canonical PSO exhibits the best performance in speed-up rate, although the dominance is not obvious.

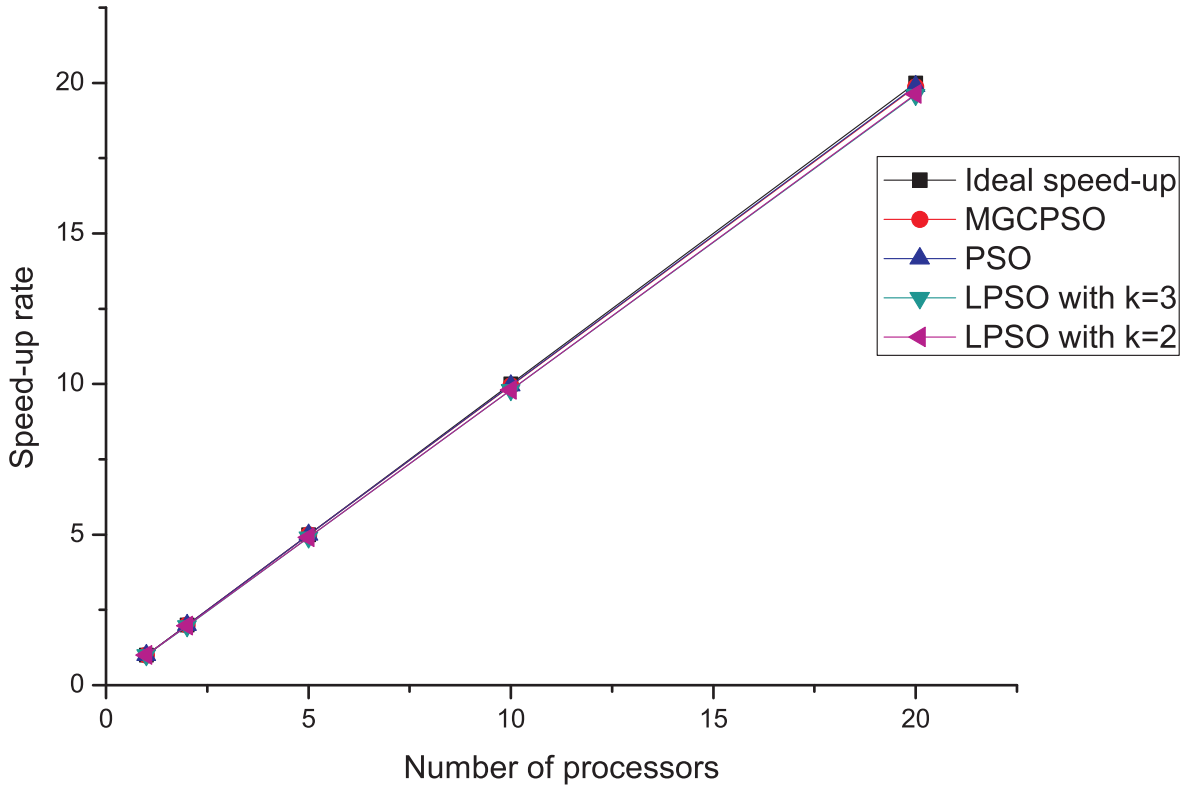


Figure 3.11: Speed-up rates

2. The LPSO has the best searching ability in the search domain, however its iteration number is extremely big, especially in the case that $k = 2$ where the iteration number is about twice of the standard PSO.

3. Although the standard PSO is the fastest algorithm its results are not very convincing because it is very easy to be trapped in a local minima without any modified mechanism.

Therefore, for usual optimization problems MGCPSO performs well but if the optimization problems are extremely difficult (e.g. high multimodal) LPSO will be the proper choice. With regard to choosing the proper topology for LPSO, there is no easy conclusion. A discussion with more examples is provide in chapter 5.

Chapter 4

Application of PSO to robust design

Traditional optimization problems only involve deterministic design parameters. In this way, many natural characteristics arising from environmental influences are neglected in the interest of easy implementation, reduced numerical effort and so on. Clearly, such models depict real-life processes, products or materials in an idealist form. In the context of structural optimization, the optima may sometimes be sensitive to even the smallest of deviations in the governing parameters. This is because any small change in these parameters can lead an active constraint to an infeasible layout. Robust Design is used to minimize the impact of variations on the system response and to increase the robustness of the designed system.

This chapter begins by presenting an overview of the concept and principles relating to robust design. Moreover, the chapter discusses the highly common meta-model technique in robust design. Further, the application of MGCPSO in the field of robust design is dealt with, as well as its parallel pattern. Finally, a conclusion on the subject matter is presented.

4.1 Robust design

4.1.1 The concept of robust design

Robust design constitutes an engineering methodology for achieving the optimal design of products, as well as that of processes which are less sensitive to system variations. It is recognized as an effective and powerful method for improving the quality of products or processes. Engineering design normally comprises three stages: conceptual design, parameter design and tolerance design. Robust design may be involved in the stages of parameter design and tolerance design.

With regard to optimization problems, the performance defined by design objectives or constraints may be subject to a large scatter at different stages of the life-cycle. Such scatter may not only have the effect of reducing quality and causing deviations from the desired performance, but may also increase life cycle costs, including those of inspection and maintenance. Well-designed products should be able to compensate for these additional costs by minimizing the effects of uncontrollable variations. In other words, excessive variations in structural performance indicates a product of poor quality. In light of these considerations,

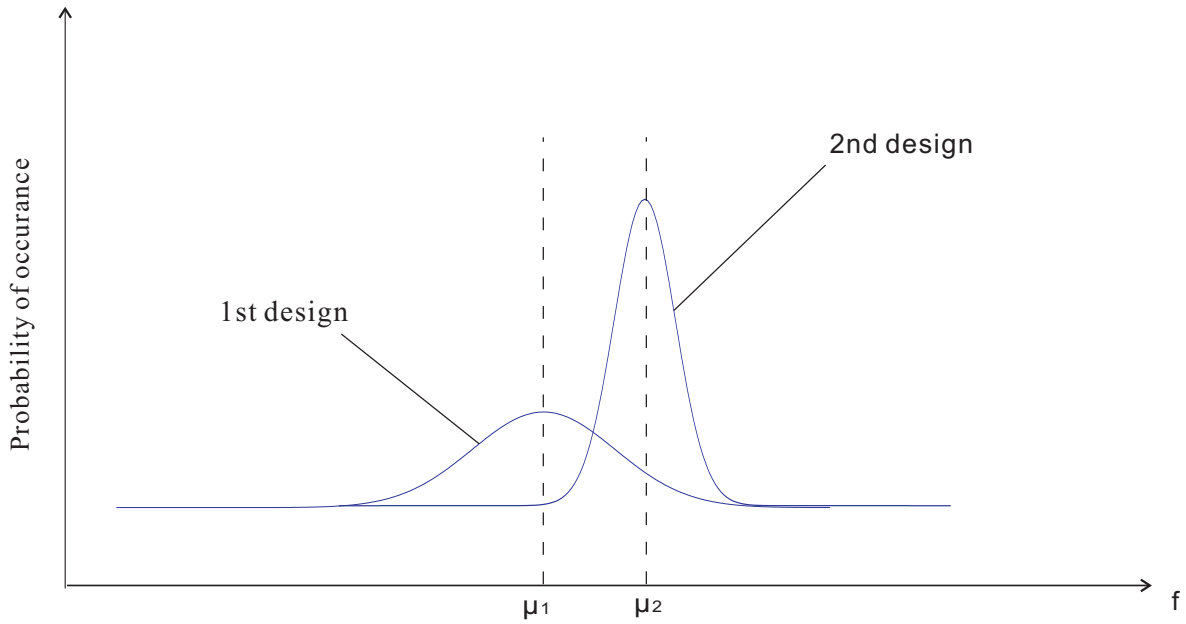


Figure 4.1: Concept of robustness

the demand for a robust design approach becomes pressing. So far, two ways of decreasing variation in structural performance have been found. One solution is to eliminate the noise parameters from the input parameters. This, however, may either be practically impossible or add - in a substantial way - to the total cost of the structure. Alternatively, a design may be sought in which the performance is less sensitive to variations in parameters, whilst not neglecting any stochastic parameters.

The basic concept of robustness is schematically shown in figure 4.1. The horizontal axis represents the value of the structural response function f, which needs to be minimized. Two curves show the probability of the value of f corresponding to two individual designs when the system parameters are randomly disturbed by the external noise parameters. In the figure, μ_1 and μ_2 represent the mean value of the fitness function f for the two designs respectively. Although the first design exhibits a small mean value of f, the second one is the more reasonable choice from a robustness point of view since it is less sensitive to variations caused by the indeterministic system parameters.

4.1.1.1 Fundamental statistical concepts

The stochastic variable z which is also known as 'noise variable' or 'random variable', exhibits stochastic properties. For instance, it cannot be assigned an exact value in each design. The Probability Density Function (PDF) ($p(z)$) and Cumulative Distribution Function (CDF) ($C(z)$) are usually used to define the occurrence properties of noise variables. In case that z is a discrete variable, $p(z)$ is often called probability function or probability mass function. In case that z is a continuous variable, $p(z)$ is normally referred to as probability density function. These two different kinds of probability function are illustrated in figure 4.2. The probability distribution function describes the probability P of a particular event occurring.

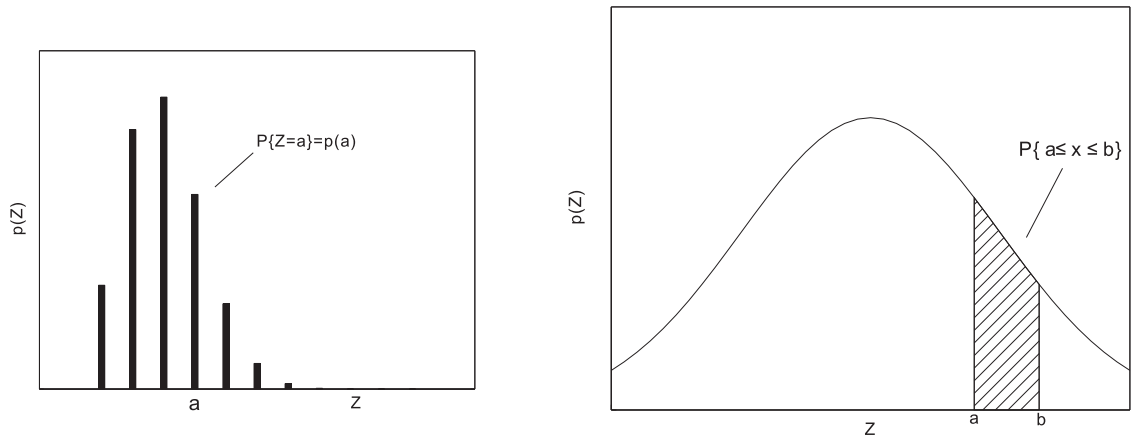


Figure 4.2: Probability functions $p(z)$ for (a) discrete and (b) continuous noise variable

It shall be noted that in the case of a discrete variable $p(z = a)$ represents the probability of the event $z = a$ whereas for continuous variable it is the integral over a range of Z (e.g. $a \leq z \leq b$) which constitutes a measure of probability. Consequently, it is meaningless to discuss the probability of the occurrence of one distinct continuous variable $z = a$. Probability distributions must show the following properties:

$$\text{discrete } z \qquad 0 \leq p(z_i) \leq 1 \qquad \forall z_i \in \Omega \tag{4.1}$$

$$P\{z = z_i\} = p(z_i) \qquad \forall z_i \in \Omega \tag{4.2}$$

$$\sum_i p(z_i) = 1 \tag{4.3}$$

$$\text{continuous } z \qquad p(z) \geq 0 \tag{4.4}$$

$$P\{a \leq z \leq b\} = \int_a^b p(z)dz \tag{4.5}$$

$$\int_\Omega p(z)dz = 1 \tag{4.6}$$

The probability distribution of the random variable z can be also be characterized by its statistical moments. There are two statistical moments which are of particular importance and they are the mean value $\mu(z)$, also known as expected value and denoted by $E(z)$, and the variance denoted by $\sigma^2(z)$ or $Var(z)$, respectively. The mean value is a measure of location (or central tendency), whilst the variance is a measure of dispersion of a probability distribution. These two functions can be defined in the following way:

$$\mu(z) = E(z) = \int_\Omega zp(z)dz \tag{4.7}$$

$$\sigma^2(z) = Var(z) = \int_\Omega (z - \mu(z))^2 p(z)dz \tag{4.8}$$

In the interest of clarity, only continuous random variables are discussed further. Another function which is very relevant to robust design is the cumulative distribution function. It

can be stated in the following terms:

$$C(z) = P\{Z \leq z\} = \int_{-\infty}^{z} p(z) \tag{4.9}$$

The function represents the probability of the event that a random realization of Z is smaller than the value z.

In structural engineering, normal and uniform distributions are among the most commonly employed types of distribution. Thus, they will be discussed briefly in the following.

Normal (or Gaussian) distribution can be written as:

$$p(z) = \frac{1}{\sigma(z)\sqrt{\pi}} \exp \frac{-(z - \sigma(z))^2}{2\sigma^2} \tag{4.10}$$

This describes the statistical behaviour of a number of natural events. A normal distribution with mean value μ and variance σ^2 is normally abbreviated as $N(\mu, \sigma^2)$. Accordingly, $x \sim N(0, 1)$ describes a random variable z that is normally distributed with $\mu(z) = 0$ and $\sigma^2(z) = 1$. Uniform distribution is denoted as:

$$p(z) = \begin{cases} \frac{1}{b-a} & a \leq z \leq b \\ 0 & z < a \vee z > b \end{cases} \tag{4.11}$$

This can be used to qualify random variables with equal occurrence of every $z \in [a, b] \subseteq \Omega$. A uniform distribution with lower and upper bound b is commonly abbreviated as $U(a, b)$. According to formula (4.7,4.8), its mean value and variance can be obtained by use of the following:

$$\mu(z) = \frac{a+b}{2} \tag{4.12}$$

$$\sigma^2(z) = \frac{(b-a)^2}{12} \tag{4.13}$$

A number of other types of distribution, including WEIBULL, POISSON and Lognormal are also highly useful in solving many structural problems that naturally arise in the field of structural design (characterization of loads, stresses, dimensions and material parameters).

It must be borne in mind that some uncertainties are not of a random nature. For this reason, they cannot be represented by means of probability distribution functions. Moreover, precise information on the probabilistic distribution of the uncertainties is often scarce or even non-existent. Therefore, non-probabilistic methods for modelling such uncertainties have been developed in recent years. However, a consideration of these innovations would be beyond the scope of this dissertation.

4.1.1.2 Formulations of robust design

The key purpose of robust design is the minimization of variation in system performance by means of involving the noise design variable and meeting the requirements of the optimum

design. The design variable can be divided into two constituents: deterministic design variable $\mathbf{x}$ and indeterministic design variable $\mathbf{z}$ (also called noise variable). The indeterministic design variable can be understood as a residual and random part of some deterministic design variables which underlies a probability distribution. Thus these design variables can be altered during the optimization process but limited into a certain precision with corresponding tolerances or probability distributions. So that the design variable of robust design $\mathbf{v}$ can be defined as:

$$\mathbf{v} = \mathbf{x} + \mathbf{z} \tag{4.14}$$

At this point in time, the orthodox optimization formula (2.1) can no longer be used in robust design because the values of objective functions and their corresponding constraints are not deterministic anymore, offering no fixed set of deterministic design variables. Instead, they show random values $\mathbf{Y}$ under a probability distribution $p(Y)$ - if the noise variable $\mathbf{z}$ is involved. In order to express the influence of the noise variable on the system's behaviour, it is necessary to establish a substitute optimization formula in which all response values are deterministic. This can be achieved by employing a statistical method, transforming the original problem with random variables into an equivalent optimization problem with only deterministic output. In this section, only the commonly used formulations are discussed.

Equality constraints with noise variables It is noted that the noise variables are able to influence the response value of equality constraints that have a fixed set of deterministic design variables. For instance, $h_k = 0$ can not be fulfilled for all $\mathbf{z} \in \Omega$. Therefore, the equality constraints should be avoided in the robust design problem. This can be done by substituting the equality requirement into the formulation of the objective and inequality constraints. If equality constraints cannot be evaded, the standard solution is to substitute the noise variable with its expected value $E(\mathbf{z})$. This is possible because the equality constraints are generally fulfilled in a mean sense and $E(\mathbf{z})$ is a statistic mean value for probability distribution functions. Two common approaches are listed below:

$$h(\mathbf{x}, \mathbf{z}) = h(\mathbf{x}, E(\mathbf{z})) = 0 \tag{4.15}$$

$$h(\mathbf{x}, \mathbf{z}) = E(h(\mathbf{x}, \mathbf{z})) = 0 \tag{4.16}$$

However, the equality will be violated for most events $\mathbf{z} \in \Omega$ which is an inherent and fundamental feature related to equality constraints that depend on random variables.

Inequality constraints with noise variables If, in robust design, the noise design variables are limited within an interval, such as $z_i \in [a, b]$, the inequality constraints could be satisfied by 100 % probability:

$$P\{g_j(\mathbf{x}, \mathbf{z}) \leq 0\} = 1 \qquad \forall \quad \mathbf{z} \in \Omega; j = 1, \cdots, n_g \tag{4.17}$$

In this case, the probability distribution of the random variables within the bounds is of no importance to the feasible domain, only the interval of z influences the solution of the optimization problem which could also be regarded as a worst-case design and is given by:

$$g_j(\mathbf{x}, \mathbf{z}) \leq 0 \qquad \forall \mathbf{z} \in \Omega; j = 1, \cdots, n_g \tag{4.18}$$

where design $\mathbf{x}$ is always feasible for all $\mathbf{z} \in \Omega$. Thus this original form of constraints is only used in case where only interval of $\mathbf{z}$ is reliably available and no information concerning the probability distribution of $\mathbf{z}$ can be obtained. Having said that, in most cases of engineering optimization it is almost impossible to determine a solution $\mathbf{x}$ which meets all the constraints perfectly. Thus, a finite probability of failure P_f can be defined for a solution $\mathbf{x}$ and is given by:

$$P_f = P\{g_j(\mathbf{x}, \mathbf{z}) > 0\} \qquad \forall \mathbf{x} \in \Omega; \quad j = 1, \cdots, n_g \tag{4.19}$$

which describes the probability of violating the constraints for a solution $\mathbf{x}$. This can be computed by an integral of the probability density function over the infeasible domain U:

$$P\{g_j(\mathbf{x}, \mathbf{z}) > 0\} = \int_U p(\mathbf{z}) d\mathbf{z}; \qquad \forall z \in \Omega; \quad j = 1, \cdots, n_g \tag{4.20}$$

where $U = \{\mathbf{z} \in \Omega \mid g_j(\mathbf{x}, \mathbf{z}) > 0\}$. Thus a new constraint can be obtained by inducing P_f and this is expressed as:

$$P\{g_j(\mathbf{x}, \mathbf{z}) > 0\} - P_{max} \leq 0 \qquad \forall z \in \Omega; \quad j = 1, \cdot, n_g \tag{4.21}$$

where the additional optimization parameter P_{max} describes the maximum probability of failure allowed. Furthermore, another design parameter P_s, called probability of safety, can be defined as:

$$P_s = P\{g_j(\mathbf{x}, \mathbf{z}) \leq 0\} = 1 - P_f \qquad \forall \mathbf{z} \in \Omega; \quad j = 1, \cdots, n_g \tag{4.22}$$

This parameter is commonly used for reliability-based design optimization (RBDO).

A more general approach of handling inequality constraints is the assessment of the cost function $\gamma(g_j(\mathbf{x}, \mathbf{z}))$ on the response value of constraints. The cost function γ is used to evaluate the violating status of inequality constraints for a certain design $\mathbf{x}$ and it should be monotonically non-decreasing i.e. $\gamma(a) \leq \gamma(b)$ for $a < b$, so that a larger violation of the constraint should be assigned equal or higher costs. Each expected value for cost of constraint violation can be restricted to a maximum allowable cost (Γ) and is given by:

$$E(\gamma_j(g_j(\mathbf{x}, \mathbf{z}))) - \Gamma_j \leq 0 \qquad \text{for each} \qquad j = 1, \cdots, n_g \tag{4.23}$$

It must be noted that the use of different approaches (such as formula (4.18, 4.21, 4.23)) results in different feasible domains.

Objective function with noise variables Analogous to traditional optimization problems, the optimum solution x^* for a stochastic optimization can be defined as:

$$f(\mathbf{x}^*, \mathbf{z}) \leq f(\mathbf{x}, \mathbf{z}) \tag{4.24}$$

where the noise has no increasing effect on the objective function and there is no other design that results in a lower objective for $\mathbf{z} \in \Omega$. The optima of the objective function in robust design is relatively difficult to achieve due to the stochastic nature of the noise variable. It must also be noted that for most engineering problems it is impossible to find optimal

robust designs $\mathbf{x} \in C$ which minimize $f(\mathbf{x}, \mathbf{z})$ for each possible realization of $\mathbf{z} \in \Omega$. In order to obtain a robust design, approximating substitute function $\rho(\mathbf{x})$ of the objective function has to be formulated in the light of statistical decision theory. This can be understood as a compromise that loosens somewhat the condition expressed in formula (4.24).

The output of the objective supplies each substitute function with a scalar value. This serves as a deterministic measurement which is used to assess the merit of each design. The decision to employ a substitute function is derived from the robust design principle. It is highly problem dependent and essentially an engineering decision.

4.1.2 Principles of robust design

In this subsection, four common principles of robust design are discussed. One of them is used in the context of this work.

4.1.2.1 Worst case scenario-based principle

In some non-deterministic structural optimization problems the design which is meant to prevent structural failure is based on the worst case analysis. In practical applications of this approach, the anti-optimization strategy is used to determine the least favourable combination of the parameter variations and thus the problem is changed to a deterministic Min-max optimization problem. This results in the evaluation of the influence of all possible events $\mathbf{z} \in \Omega$ on each design variable x. Also, the fitness value derived from the worst case is used to measure the robustness of the current design. Therefore no probability density function of the noise variables are required, the corresponding substitute function can be defined as:

$$\rho(\mathbf{x}) = \max_{\mathbf{z}} f(\mathbf{x}, \mathbf{z}) \qquad \forall \mathbf{z} \in \Omega \tag{4.25}$$

the task of robust design is to optimize $\rho(\mathbf{x})$. This principle ignores the probability distribution of noise variables. Instead, it specifies the upper limit of the resulting range and thus leads to very conservative designs. Elishakoff et al. [40] applied this method to the issue of bounded uncertainty, arising in the context of a structural optimal design problem. Yoshikawa et al. [140] created a concept which allows for the computing of the worst case scenario with regard to homology design. It draws from the uncertain fluctuation of loading conditions and uses the convex uncertainty model. The validity of the proposed method can be demonstrated by applying it to the design of simple truss structures. Lombardi and Haftka [84] combined the worst case scenario technique of anti-optimization and the structural optimization techniques to the structural design and applied them to structural design under uncertainty.

4.1.2.2 Quantile based principle

In the case of this principle, the substitute function $\rho(\mathbf{x})$ is the inverse of the cumulative distribution function for the random objective with an addition parameter (a quantile q (e.g.

90% quantile)) and is given by:

$$\rho(\mathbf{x}) = C^{-1}(\mathbf{x}, q) \tag{4.26}$$

This function can also be considered as a function of the design variables $\mathbf{x}$ because in general the probability density p of the response also depends on the design variables. In the case of a robust design $\mathbf{x}^*$, the relevant percentage of possible events $\mathbf{z} \in \Omega$ will result in a value of objective $(f(\mathbf{x}, \mathbf{z}))$ that is smaller than or equal to the value of $(f(\mathbf{x}^*, \mathbf{z}))$. It is noted that if this principle is evaluated for $q = 1$, the resulting substitute function $\rho(\mathbf{x})$ for the uniform distribution is equal to the minimax principle.

4.1.2.3 Cost function based principle

Analogous handling inequality constraints with random variables, this principle employs the cost function γ to formulate the substitute function $\rho(\mathbf{x}, \mathbf{z})$, thereby assigning different costs to the differing response values $f(\mathbf{x}, \mathbf{z})$ and $\mathbf{g}(\mathbf{x}, \mathbf{z})$. In this way, the expected overall cost for the violation of constraints, as well as any deviations in the objective can be optimized.

$$\rho(\mathbf{x}) = E(\gamma_0(f(\mathbf{x}, \mathbf{z}))) + \sum_{j=1}^{n_g} E(\gamma_j(g_j(\mathbf{x}, \mathbf{z}))) \tag{4.27}$$

The advantage of the cost function principle is that different elements of the stochastic optimization problem (violence of each constraint, mean and variance of the objective) is combined into a consistent basis. Meanwhile, the problem usually becomes an unconstrained optimization problem since all constraints are presented in terms of their equivalent and are summarized in the new objective function, namely the overall costs. A problem which adopts this approach is able to define appropriate and accurate cost functions for each possible contribution. The difficulty entailed in the use of this principle is that it requires the choosing of proper cost functions for all the different elements so that the total cost can describe the performance of a certain solution more accurately.

4.1.2.4 Multi-criteria based principle

This principle treats robust optimization as a multi-criteria optimization problem with two objectives: to minimize the location, as well as the dispersion of the random response, for instance the mean and the variance. The easiest and most straightforward way to construct the substitute function $\rho(\mathbf{x})$ is to assign different user-specified weighted factors on the mean and the variance of the objective and summarize them to achieve a uniform value.

$$\rho(\mathbf{x}) = \omega_1 \mu(f(\mathbf{x}, \mathbf{z})) + \omega_2 \sigma(f(\mathbf{x}, \mathbf{z})) \tag{4.28}$$

Another very common variance of $\rho(\mathbf{x})$ - known as signal to noise ratio (SNR) principle - is given by:

$$\rho(\mathbf{x}) = 10 \ \log_{10}(\mu(f(\mathbf{x}, \mathbf{z}))^2 + \sigma(f(\mathbf{x}, \mathbf{z}))^2) \tag{4.29}$$

This variance was first introduced by Taguchi [127] where zero is assumed to constitute the minimal response value possible. The manipulator $10\ \log_{10}()$ is used to transform the magnitude of the robustness criterion into decibel units (dB) and it will not change the optimum position.

In essence, $\rho(\mathbf{x})$ is needed to characterize the optimum in order to achieve the objectives of minimizing the location, as well as the dispersion of the random response.

4.2 Apply Metamodel to robust design

Today, most engineering design problems rely, to an extreme extent, on numerical experiments (know as computational simulation) for the evaluation of the system's response. However, in this way, only discrete information about the underlying relationship can be obtained. To illustrate this, if we want to obtain information about the structural response - concerning, for example, deformation or member stress - for a certain design, a finite element analysis is commonly employed to gain this information. Having said that, it is possible to achieve an analytical solution, albeit only in the case of very simple structures, including the classic 3-bar truss. In general, in the case of a lot of the real world problems, a single simulation can take a vast amount of time to complete. This is particularly so with regard to non-linear analysis. Due to the need for a great many simulation evaluations, even routine tasks such as design optimization, design space exploration, sensitivity analysis, what-if analysis and stochastic optimization become virtually impossible to execute. For instance, a sub-optimization task is nested in order to obtain the worst case for a certain design $\mathbf{x}$ in the case of robust design based on Max-Min principle. This task has to be repeated for each intermediate design $\mathbf{x}$ until a robust design $\mathbf{x}^*$ is achieved. This is a really time-consuming procedure and makes stochastic optimization of the original problem prohibitive. For these reasons, it is highly unlikely that problems of stochastic optimization will become easy to handle.

One way of dealing with this problem is to construct approximate models (also called surrogate models, response surface models, metamodels or emulators) that can mimic the behavior of the simulation model as closely as possible. One of the main advantage of this approach is that it allows for evaluation by computer which is relatively cheap. The metamodel $\hat{f}(\mathbf{x})$ is a set of mathematical functions and it is usually constructed with regard to the responses obtained from the simulator which focuses on a limited number of intelligently chosen data points through design of experiment (DoE). The initial system responses are obtained by using some of the budget available for more expensive experiments and simulations. Then the surrogate model can predict the remaining experiments and simulations which are run to create designs and may demonstrate a dominating performance. In case there is only one design variable involved, the process is known as curve fitting. This notion has been successfully applied in the field of robust design. To illustrate this, Florian Jurecka [70] implemented it to the auto mobile industry. His approach entails the setting up of a surrogate model $\hat{f}$ with selected design variables $\mathbf{x}$, noise variables $\mathbf{z}$ and corresponding response values y. In the interest of simplicity, variable $\mathbf{v}$ is used as a combination of design variables $\mathbf{x}$ and noise variables $\mathbf{z}$. Thus, the object function $f(\mathbf{x}, \mathbf{z})$ is changed to $f(\mathbf{v})$.

In this work, for the construction of a meta model, the Kriging model and the Latin-Hyper-Cube are employed for the definition of a modelling technique, as well as the supply of sampling points. Normally, the initial model is incapable of accommodating the original function perfectly so that the need for an updating method arises. One answer is to insert additional sampling points.

4.2.1 Kriging model

The most popular surrogate models are polynomial response surfaces, Kriging model and artificial neural networks. For most problems, the nature of the function cannot be known a priori so that it is difficult to know which surrogate will offer the most accurate information. In this research work, Kriging model is used to construct an approximate model which is very flexible in nature due to the fact that it allows for a large variety of feasible correlation functions.

The theory of Kriging was developed by the French mathematician Matheron [87] and it is based on the Master's thesis of Krige [75]. The latter was a pioneer in the field of plotting distance-weighted average gold grades at the Witwatersrand reef complex. The Kriging theory now comprises a number of geo-statistical techniques which are used to interpolate the value of a random field at the location under investigation. This is done by observing its value at nearby locations and by employing a widespread global approximate technique. Sacks et al. [115] have first coded this approach to model deterministic output, and this model is called DACE which is the abbreviation of their contribution - "Design and Analysis of Computer Experiments". The basic function of Kriging model $\hat{y}$ is given by:

$$y \approx \hat{y} = \eta(\mathbf{v}, \boldsymbol{\beta}) + Z(\mathbf{v}) \tag{4.30}$$

where $\eta(\mathbf{v}, \boldsymbol{\beta})$ and $Z(\mathbf{v})$ are regression and correlation parts respectively. $\eta(v, \beta)$ is normally a set of polynomials $\boldsymbol{\eta}(\mathbf{v}) = [\eta_1(\mathbf{v}), \eta_2(\mathbf{v}), \cdots, \eta_{n\beta}(\mathbf{v})]^T$ with regression parameters $\boldsymbol{\beta}$ which gives a global trend for system response. Depending on the regression functions selected, the Kriging technique can be classified into different categories:

1. Simple Kriging assumes a known constant trend model.
2. Ordinary Kriging assumes an unknown constant trend model.
3. Universal Kriging assumes a general linear trend model.
4. Lognormal Kriging interpolates positive data by means of logarithms.
5. And so on...

$Z(\mathbf{v})$ is a random process and can be assumed to be a normally distributed Gaussian Random process with mean zero, variance σ^2 and covariance Cov. This can be thought of as an interpolation of the residuals of the regression models $\eta(\mathbf{v}, \boldsymbol{\beta})$, thus the residuals at the sampling points vanish $\mathbf{e}_l = \mathbf{y}^l - \hat{\mathbf{y}}^l = 0; \quad l = 1, \cdots, n$. The covariance is a product of σ^2 and the correlation function between $Z(\mathbf{v^l})$ and $Z(\mathbf{v}^k)$ and is given by:

$$Cov(\mathbf{v}^l, \mathbf{v}^k) = \sigma^2 R(\mathbf{v}^l, \mathbf{v}^k) \tag{4.31}$$

In this work, the correlation functions are restricted in the following way:

$$R(\mathbf{v}^l, \mathbf{v}^k) = \prod_{i=1}^{m} R_i(v_i^l, v_i^k) \tag{4.32}$$

i.e. they are the products of m (size of vector $\mathbf{v}^l$) one-dimentional correlation functions $R_i(v_i^l, v_i^k)$. Suitable choice of one-dimensional correlation function can be e.g.
the Gaussian correlation function

$$R_i(v_i^l, v_i^k) = \exp\left(-\theta_i (v_i^l - v_i^k)^2\right) \tag{4.33}$$

the linear correlation function

$$R_i(v_i^l, v_i^k) = \max\left\{0, 1 - \theta_i \mid v_i^l - v_i^k \mid\right\} \tag{4.34}$$

and the cubic correlation function

$$R_i(v_i^l, v_i^k) = 1 - 3\epsilon_i^2 + 2\epsilon_i^3 \qquad \epsilon_i = \min(1, \theta_i \mid v_i^l - v_i^k \mid) \tag{4.35}$$

The selection of the correlation function is normally dependent on the underlying phenomenon, e.g. a function to be optimized or a physical process to be modeled. If the underlying phenomenon is continuously differentiable, the correlation function should show parabolic behavior near the origins which would ordinarily suggest that a linear correlation function is the better choice. Interestingly, the physical phenomenon shows linear behavior near the origins, linear correlation function would normally perform better. The parameter θ in formula (4.33,4.34,4.35) is called the correlation parameter which controls the range of influence of the sampling points and has to be a positive constant. Therefore, the correlation function in formula (4.32) can be expressed as:

$$R(\boldsymbol{\theta}, \mathbf{v}^l, \mathbf{v}^k) = \prod_{i=1}^{l} R_i(\theta_i, \mathbf{v}_i^l, \mathbf{v}_i^k) \tag{4.36}$$

if correlation parameter θ is treated as an additional parameter.

From the responses obtained in the aforementioned way, the known variables are sampling points $\mathbb{V} = \{\mathbf{v}^1, \mathbf{v}^2, ..., \mathbf{v}^n\}$ and their corresponding system responses $\mathbf{y} = [y_1, y_2, ..., y_n]^T$. From these responses, the unknown parameters β and σ^2 of Kriging model can be estimated by:

$$\boldsymbol{\beta} \approx \hat{\boldsymbol{\beta}} = (\mathbf{F}^T \mathsf{R}^{-1} \mathbf{F})^{-1} \mathbf{F}^T \mathsf{R}^{-1} \mathbf{y} \tag{4.37}$$

$$\sigma^2 \approx \hat{\sigma^2} = \frac{1}{n} (\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}})^T \mathsf{R}^{-1} (\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}}) \tag{4.38}$$

where $\mathbf{F}$ and R are the regression vector (defined in formula 4.39) and the correlation matrix (4.40) respectively (more details about how to derive these formulas can be found in [70]).

$$\mathbf{F} = \begin{bmatrix} \eta_1(\mathbf{v}^1) & \eta_2(\mathbf{v}^1) & \cdots & \eta_{n\beta}(\mathbf{v}^1) \\ \eta_1(\mathbf{v}^2) & \eta_2(\mathbf{v}^2) & \cdots & \eta_{n\beta}(\mathbf{v}^2) \\ \vdots & \vdots & & \vdots \\ \eta_1(\mathbf{v}^n) & \eta_2(\mathbf{v}^n) & \cdots & \eta_{n\beta}(\mathbf{v}^n) \end{bmatrix} \tag{4.39}$$

$$\mathsf{R} = [R(\theta, \mathbf{v}^i, \mathbf{v}^j)]_{i,j} \qquad 1 \le i, j \le n \tag{4.40}$$

Using these estimated parameters, the best linear unbiased prediction can be written as:

$$\hat{y} = \mathbf{f}(\mathbf{v})^T\hat{\boldsymbol{\beta}} + \mathbf{r}^T(\mathbf{v})\mathsf{R}^{-1}(\mathbf{y} - F\hat{\boldsymbol{\beta}}) \tag{4.41}$$

with the correlation vector

$$\mathbf{r}^T(\mathbf{v}) = \left[R(\mathbf{v},\mathbf{v}^1), R(\mathbf{v},\mathbf{v}^2), \cdots, R(\mathbf{v},\mathbf{v}^n)\right] \tag{4.42}$$

which is a column of the individual correlations between the current prediction point $\mathbf{v}$ and each sampling point $\mathbf{v}^i$.

4.2.2 Latin Hypercube Sampling

The modelling technique discussed in subsection 4.2.1 depends on training data is obtained from sampling points. Because a metamodel is a numerical surrogate model of the original problem, the quality of the sampling points is capable of having an effect on the accuracy of the meta model, albeit only to a limited extent. The 'Design of Experiments' (DoE) technique is used for selecting the appropriate sampling points in a given design space. In essence, the method relies on picking sampling points of a quantity that is suitable for gathering the maximum amount of information possible on the main features of the system under investigation. The great advantage of DoE is that it provides an organized approach through which it is possible to address both simple and tricky problems[43]. Another advantage of DoE is that it can eliminate or decrease the influence of noise parameters - which may or may not be recognized - and would otherwise affect the fundamental relationship between the controlled input and system's response values[70]. However, since it is the express goal of robust design to explore the nature of the influence of noise parameters, as well as to create a system that is insensitive to them, it is essential to explicitly involve noise parameters in the formulation of the problem and the deterministic design. Thus, it is of utmost significance in the field of robust design to hold the noise variables in a controllable interval during the stages of engineering and design.

The application of "design of experiments" essentially entails the running of a set of n experiments that are expressed in terms of the input variable $\mathbf{v}$ which contains the design variables and all controllable noise parameters. The dissertation of Florian Jurecka [70] contains detailed descriptions of the many different sampling methods. Latin Hypercube Sampling LHS is used in this work because it constitutes an attractive method for constructing the experimental design.

This statistical method was initially developed to generate a distribution of plausible collections of parameters in a multidimensional design space by McKay [88] in 1979. It was then elaborated by Iman et el. [67] in 1981. After approximately twenty years of development, the LHS has been successfully used to generate multivariate samples of statistical distribution and applied to many different computer models.

In the context of statistical sampling, the term Latin Square refers to a square grid where there is only one sampling point in each row and each column. In a multi-dimensional case, this means that each axis-aligned hyperplane contains only one sample. A latin hypercube

is the generalization of this concept. In order to obtain M sampling vector $\mathbf{v}^i$ with dimension N, the range of each dimension of variable $\mathbf{v}_i$ is divided into M equally probable intervals. The M sampling vectors are placed into these grids to satisfy the latin hypercube requirements. It is important that the number of divisions M must equal the required number of vectors. The total number of combinations of the sampling points for a Latin Hypercube of M divisions and N dimensions can be evaluated by use of the following formula:

$$\prod_{n=0}^{M-1} (M-n)^{N-1} \tag{4.43}$$

For instance, if $M = 4$ and $N = 2$ (e.g. a square), the resulting LHS has 24 potential combinations. If $M = 4$ and $N = 3$ (e.g. a cube), the resulting LHS has 576 potential combinations. This is a large variety, especially for the sampling points' quality. In order to improve the quality of the sampling points, a criterion- knows as the maximin distance criterion - was proposed to maximize the distance between an arbitrary prediction point and its closest sampling point. However, it is a complex approach since an infinite number of designs have to be evaluated. The standard way of reducing the vastness of the computational effort is to reduce the number of candidate combinations by means of another criterion, including randomized orthogonal arrays in [98], nested design [55], minimum potential energy [10].

4.2.3 Iterative model update technique

There will always be a discrepancy between the predicted and the true response as long as the metamodel is used as a surrogate for the original computer simulation. Typically, these metamodels are used for robust design so that the response value on the points in the design space are obtained using the prediction of the metamodel. As a result of the robust design, a predicted optimum is obtained which can be seen as a approximation of the true optimum. One way of increasing the reliability and accuracy of the predicted optimum is to improve the quality of the sampling points. The fact that there exists a range of ways for improving the quality of sampling points in a latin hypercube doesn't change the reality that initial points normally cannot result in a satisfactory metamodel. For this reason, additional points need to be included iteratively in the design space. A sequential approach is adopted in this work so that the effect of the metamodel can be increased by an iterative update procedure.

An approach called efficient global optimization (EGO) was first proposed by Jone et al. [69] to achieve faster convergence on the global optimum with metamodels. The method balances out the global and local search based on metamodels which are iteratively updated during the optimization process. In this method, local search means a detailed investigation of the behavior around the predicted optimum. Meanwhile, global search denotes the elimination of the possibility of missing the optimum due to an error in prediction. These two goals need to be weighed up when looking for new points for filling in. This approach lends itself very well to optimization that relies on metamodels. However, all the variables used in this method are deterministic. In order to apply this approach to robust design using metamodels, Florial Jurecka in [70] proposed a suitable modification of the standard

EGO to treat random variables in the problem. In his approach, the search for infill points is split into two parts. First, the design space is explored to find design variables $\mathbf{x}'$ which are most promising with respect to the robustness formulation $\rho(\mathbf{x})$. Secondly, the noise space is explored so that a proper noise variable $\mathbf{z}'$ is obtained. $\mathbf{x}'$ and $\mathbf{z}'$ are then combined to form the desired infill point $\mathbf{v}'$. Following the discovery of the infill point, the original system response on this point can be obtained, allowing for the metamodel to be updated. In the following, the aforementioned model update procedure shall be discussed in brief. The detailed description can be found in [70]. Before discussing the approach in detail, some symbols shall be defined.

- ◇ Mean squared error(MSE) is used as the estimated prediction variance of the kriging predictor $\hat{y} = \hat{y}(\mathbf{v})$ and is given by

$$MSE(\hat{y}(\mathbf{v})) = \sigma^2 \left(1 - \begin{bmatrix}\boldsymbol{\eta}^T(\mathbf{v}) & \mathbf{r}^T(\mathbf{v})\end{bmatrix} \begin{bmatrix}\mathbf{0} & \mathbf{F}^T \\ \mathbf{F} & \mathsf{R}\end{bmatrix} \begin{bmatrix}\boldsymbol{\eta}(\mathbf{v}) \\ \mathbf{r}(\mathbf{v})\end{bmatrix}\right) \tag{4.44}$$

- ◇ response value $\tilde{y}^*$ means the best tried and proven choice based on the response values y_l of sampling points $\mathbf{x}^l$(with $l = 1, ..., n$) and is given by

$$\tilde{y}^* \leq y_l \qquad \forall \quad l \in \{1, 2, \cdots, n\} \tag{4.45}$$

- ◇ improvement I over $\tilde{y}^*$ related to an arbitrary y is given by

$$I = \max\{0, (\tilde{y}^* - y)\} \tag{4.46}$$

- ◇ Expected value of I

$$E(I) = E(\max\{0, (\tilde{y}^* - y)\}) \tag{4.47}$$

Design Space Exploration The criterion to find the design variable part $\mathbf{x}'$ is to maximize the expected improvement defined in formula (4.47). The estimated prediction variance $\hat{s}^2$ can be approximated by maximizing the MSE over all events $\mathbf{z} \in \Omega$ and is given by

$$\hat{s}^2 = \max_{\mathbf{z}\in\Omega} MSE(\mathbf{x}, \mathbf{z}) \tag{4.48}$$

This means that a large MSE value anywhere in the noise space affects greatly the accuracy of the robustness criterion in terms of prediction. Since $\hat{y}$ is the prediction of the system response at $\mathbf{x}$ computed through a metamodel to simulate the real value y, after the value of $\hat{s}$ is obtained, the expected improvement $E(I)$ can be expressed in close form as:

$$E(I) = \int_{-\infty}^{\tilde{y}^*} (\tilde{y}^* - y) p y(y) dy = (\tilde{y}^* - \hat{y})\Phi\left(\frac{\tilde{y}^* - \hat{y}}{\hat{s}}\right) + \hat{s}\phi\left(\frac{\tilde{y}^* - \hat{y}}{\hat{s}}\right) \tag{4.49}$$

where ϕ is the probability density function of the standard normal distribution $N(0, 1)$ and Φ is the cumulative density function of the same distribution. Moreover, formula (4.49) can

be expressed in terms of y_1 and y_2 which are intersection points between two probability density functions of p_Γ and p_{Γ^*} and are evaluated by

$$y_{1,2} = \frac{\hat{s}^2\hat{y}^* - (\hat{s}^*)^2\hat{y} \pm \hat{s}\hat{s}^*\sqrt{(\hat{y}-\hat{y}^*)^2 + 2\ln\left(\frac{\hat{s}}{\hat{s}^*}\right)(\hat{s}^2-(\hat{s}^*)^2)}}{\hat{s}^2-(\hat{s}^*)^2} \tag{4.50}$$

Detailed derivations of these formulas can be found in [70]. It is noted that the probability density functions are according to normal distributed function, thus the random numbers obtained from p_Γ and p_{Γ^*} are characterized by $\Gamma \sim N(\hat{y},\hat{s})$ and $\Gamma^* \sim N(\hat{y}^*,\hat{s}^*)$ respectively. For consistence, the lower intersection value is always denoted by y_1, the upper point by y_2, i.e. $y_1 \leq y_2$. Based on different values of y_1 and y_2, the approximation of $E(I)$ is

1. if $\hat{s} > \hat{s}^*$

$$E(I) = (\hat{y}^* - \hat{y})\Phi\left(\frac{y_1-\hat{y}}{\hat{s}}\right) + \hat{s}\phi\left(\frac{y_1-\hat{y}}{\hat{s}}\right) - (\hat{s})^*\phi\left(\frac{y_1-\hat{y}^*}{\hat{s}^*}\right) \tag{4.51}$$

2. if $\hat{s} = \hat{s}^*$, $E(I)$ can also be evaluated using formula (4.51). The only difference is that in this case the lower point is identified as $y_1 = (\hat{y}^* + \hat{y})/2$

3. if $\hat{s} < \hat{s}^*$, there are three different cases to be considered

   (a) if $y_1 < \hat{y}^*$ and $y_2 < \hat{y}^*$

$$\begin{aligned} E(I) = {} & (\hat{y}^* - \hat{y})\left(\Phi\left(\frac{y_2-\hat{y}}{\hat{s}}\right) - \Phi\left(\frac{y_1-\hat{y}}{\hat{s}}\right)\right) \\ & + \hat{s}\left(\phi\left(\frac{y_2-\hat{y}}{\hat{s}}\right) - \phi\left(\frac{y_1-\hat{y}}{\hat{s}}\right)\right) - \hat{s}^*\left(\phi\left(\frac{y_2-\hat{y}^*}{\hat{s}^*}\right) - \phi\left(\frac{y_1-\hat{y}^*}{\hat{s}^*}\right)\right) \end{aligned} \tag{4.52}$$

   (b) if$y_1 < \hat{y}^*$ and $y_2 > \hat{y}^*$

$$\begin{aligned} E(I) = {} & (\hat{y}^* - \hat{y})\left(\Phi\left(\frac{\hat{y}^*-\hat{y}}{\hat{s}}\right) - \Phi\left(\frac{y_1-\hat{y}}{\hat{s}}\right)\right) \\ & + \hat{s}\left(\phi\left(\frac{\hat{y}^*-\hat{y}}{\hat{s}}\right) - \phi\left(\frac{y_1-\hat{y}}{\hat{s}}\right)\right) - \hat{s}^*\left(\frac{1}{\sqrt{2\pi}} - \phi\left(\frac{y_1-\hat{y}^*}{\hat{s}^*}\right)\right) \end{aligned} \tag{4.53}$$

   (c) if$y_1 > \hat{y}^*$ and $y_2 > \hat{y}^*$

$$E(I) = 0 \tag{4.54}$$

In summary, when the parameters $\hat{y}^*$,$\hat{s}^*$,$\hat{y}$ and $\hat{s}$ are known an optimization method can be to find the maxima of $E(I)$ and obtain the design variables $\mathbf{x}'$. It is noted that, during the procedure to search $\mathbf{x}'$, $\hat{y}^*$ and $\hat{s}^*$ are only calculated once, while $\hat{y}$ and $\hat{s}$ have to be evaluated for each candidate point.

Noise Space Exploration Once the most promising design variables $\mathbf{x}'$ are obtained, the noise space needs to be explored in order to find the $\mathbf{z}'$. The rule which is followed in identifying $\mathbf{z}'$ depends on the robustness criterion ρ. Because in this work the worst case scenario is used as the robustness criterion, the goal at this optimization stage is to find the worst case $y^{\#}$ in order to maximize the deteriorating effect of the noise variables. Therefore, the objective function at this stage should represent a trade-off between worsening objective and reducing the prediction error of the model in the noise space. The prediction error $\hat{s}$ at any point $\mathbf{z}$ can be computed as $\hat{s} = \sqrt{MSE(\mathbf{x}', \mathbf{z})}$. Normally, we can not know the reference value $y^{\#}$ in advance for which the prediction error vanished. Therefore, we need to find $\hat{y}^{\#}$ with the corresponding noise variable $\hat{z}^{\#}$ which has the smallest prediction error $s^{\#}$ as a replacement of $y^{\#}$. For further work, a new parameter W is defined as:

$$W = \max_{\mathbf{z}\in\Omega} \left\{0, \left(y - y^{\#}\right)\right\} \approx \max_{\mathbf{z}\in\Omega} \left\{0, \left(y - \hat{y}^{\#}\right)\right\} \tag{4.55}$$

which presents the worsening of the objective. The expected value of worsening W can be defined as:

$$\begin{aligned} E(W) &= E\left(\max\left\{0, \left(y - y^{\#}\right)\right\}\right) \\ &= \left(\hat{y} - \hat{y}^{\#}\right)\left(1 - \Phi\left(\frac{y_2 - \hat{y}}{\hat{s}}\right)\right) + \hat{s}\phi\left(\frac{y_2 - \hat{y}}{\hat{s}}\right) - \hat{s}^{\#}\phi\left(\frac{y_2 - \hat{y}^{\#}}{\hat{s}^{\#}}\right) \end{aligned} \tag{4.56}$$

where y_2 can be obtained by using formula (4.50). Consequently, similar to the procedure used in design space exploration, the task of finding $\mathbf{z}'$ involves determining the maximum of the expected value $E(W)$ defined in formula (4.56). As a result, the noise variable part $\mathbf{z}'$ of the infill point is obtained. The new infill point $\mathbf{v}'$ is then achieved by combining $\mathbf{x}'$ and $\mathbf{z}'$. This procedure will be continued, until either a lower bound on $E(I)$ of the design or the maximum number of updating procedure is reached.

4.3 Apply MGCPSO to robust design with Kriging model

Since the expected improvement function is multimodal, it is necessary to choose a global optimization algorithm to obtain the proper infill points, such as genetic algorithm or particle swarm optimization. In chapter 3, a new modified optimizer call modified guaranteed convergence particle swarm optimizer (MGCPSO) is proposed. It gains a trade-off between global search ability and computing efficiency in benchmark test compared with standard particle swarm optimizer and a modified **Lbest** based particle swarm optimizer (LPSO). For this reason, this approach (MGCPSO) is integrated into the iterative metamodel update procedure for obtaining infill points. As discussed in subsection 4.2.3, three sub-optimization problems are nested in each sub-step in finding infill points gives rise to three optimization problems. They are finding $\mathbf{x}'$ by maximizing the expected value of improvement $E(I)$, finding reference noise part $\hat{\mathbf{z}}^{\#}$ by minimizing the prediction error $\hat{s}$ and finding $\mathbf{z}'$ by maximizing the expected value of the worsening $E(W)$ respectively. As a consequence, in each update iteration MGCPSO is used for the solving of the optimization problems. As a result, the infill point is found.

Normally, the number of infill points cannot be known in advance. Solving three sub-optimization problems in order to obtain one infill point is a relatively expensive endeavour from at computing point of view. This effect is obviously increasing as the number of infill points is increasing, because using more sampling points will cause a more complex meta-model which fits the original system better. This, in turn, will increase the complexity of predicting system response from metamodels on investigating points. Moreover, in the case of the objective of the original system being highly multi-modal, a meta model that fits better will be problematic with regard to the integrated optimum, meaning that the optimizer would require more loops to identify the global optimizer. Therefore, parallel computing is taken into account. Another major reason for using MGCPSO here is that it is scalable in parallel computing and has an efficient speed-up ratio in a benchmark test..

The research in this chapter can be seen as an extension of Florian Jurecka's research. His working platform is mainly Matlab and the Kriging model part of his program is based on DACE (Design and Analysis of Computer Experiments) which is a reliable and wildly used Matlab toolbox for working with Kriging approximations to computer models and developed by Hans Bruun Nielsen and et al. from Technical University of Denmark. Meanwhile the parallelizing MGCPSO is written in Fortran. In order to reuse the current code and save working effort as much as possible, the program in this work is partly written in Matlab code. The main reason why it is not written completely in Matlab code is that Matlab dose not support parallel computing very well and parallel computing by means of Matlab is not yet well studied. Besides, the current methods available for parallizing the Matlab program in a distributing environment require additional Matlab licenses. In order to run the Matlab application on all computing nodes simultaneously, it is necessary to install Matlab on each computing node. Thus, as many licenses as there are computing nodes are required which constitutes a genuine financial burden to the research institute. This, in turn, restricts the possible scale at which parallel computing can be executed. Fortunately, Matlab provides an interface to Fortran, C and C++ , both of which are very convenient for parallel computing without incurring any additional software cost. Hence, the mixed use of both Matlab and Fortran constitutes a very good compromise for handling the aforementioned problem. Mixed programming is advantageous for a number of reasons. First of all, it allows for the re-use of current codes which is consistent with one of the goals of modern programming. Moreover, Fortran is more efficient at computation than Matlab, so the use of Fortran in part, but particularly when in parallel mode, can decrease the computing effort significantly.

In this approach, master-slave mode is used as the parallel pattern which coincides with the parallelizing MGCPSO discussed in subsection 3.3.2. As a result, only one copy of Matlab is installed on each computing node. The DACE comprises two parts: Constitution part and prediction part. The constitution part is executed only in case of building or updating metamodel after infill points are obtained. This part is also the most difficult part of DACE, so that it is not necessary to rewrite this part in Fortran but instead, keep it in Matlab code. However, the prediction part is invoked frequently during optimization procedures, because the system response needs to be approximated using a surrogate model for each intermediate point. Therefore, it is crucial for the prediction to be re-written in Fortran. The parallelizing MGCPSO is used to solve the three sub-optimization problems which arise in each updating iteration. The master computing node is responsible for constituting meta-

model, as well as organizing the parallelizing optimization tasks. The slave nodes, on the other hand, are in charge of executing the corresponding optimization part, albeit under the control of the master node. This task includes the prediction of system responses by using metamodels through Fortran codes. The following shows in detail how MGCPSO can be applied to robust design through the use of metamodel and mixed programming. Firstly, the master node generates a certain number of sampling points $\mathbf{x}$,observes the system response $\mathbf{y}$ at $\mathbf{x}$ and creates the kriging model with these informations. These tasks are done in Matlab. After the kriging model is created, the information necessary for predicting system response by way of the kriging model are scattered to slave nodes for further computations. This is done with Fortran codes. Then, the parallelizing MGCPSO is applied to solve sub-optimization problems to obtain infill points $\mathbf{v}'$ which is done also with fortran codes. Hereafter, the master node updates the kriging model with original points and $\mathbf{v}'$, which is done with Matlab. Then the master node scatters the information of the updated kriging model to slave nodes to replace their out-dated information and the next new infill node needs to be searched for through parallelizing MGCPSO. This is done with Fortran codes. Actually, after obtaining $\mathbf{x}'$ by maximizing the expected value of improvement $E(I)$, the maximal value of $E(I)$ can be used to determine whether the stopping criterion is met or not. If the stopping criterion is not met, program will continue to find the new infill node through parallelizing MGCPSO. When the stopping criterion is met, the program will stop and output the final kriging model for further research, e.g. robust design. The flowchart is presented in figure 4.3. In the next section, a benchmark test is implemented to evaluate the performance of this approach compared with DIRECT.

4.4 Numerical experiments and Conclusion

In this section, in order to evaluate the performance of MGCPSO when applied to robust design problems, two classic robust design problems selected from [70] are solved by means of MGCPSO. Solutions are compared with those obtained by DIRECT [48]. The essential characteristics for robust design are given by:

1. The kriging model is used to simulate the original test function.
2. The regression function is first order polynomial.
3. The gauss function is chosen as the correlation function.

The parameter set for PSO is given by

1. **Inertia Weight** ω: This coefficient is used to control the trajectories of particles and is set to $\omega = 0.5 + rand()$.
2. **Acceleration Coefficients** φ_1 and φ_2: These are mostly used in the community of particle swarms, the values are $\varphi_1 = \varphi_2 = 1.49445$.
3. **Maximum Velocity** v_{max}: This parameter was not set, as it was not deemed necessary.

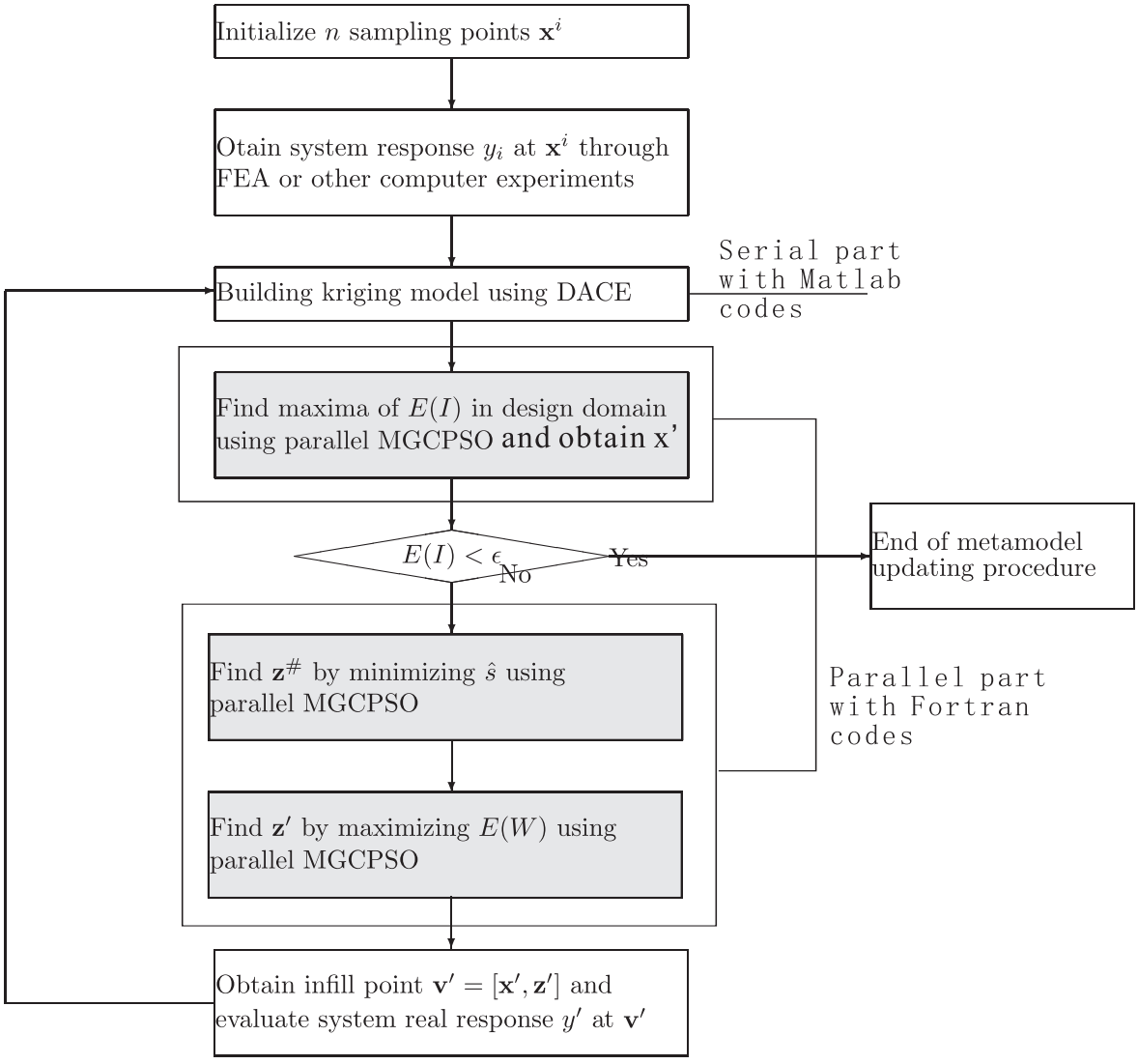


Figure 4.3: Workflow of applying parallel MGCPSO to robust design with kriging model

4. **Population Size**: All the swarms used in this study comprise ten individuals.

5. **Stopping criterion**: If the best position of swarm can not be improved in fifty consecutive iterations, the program will be stopped artificially and the fitness of the best position will be considered as the result of this numerical test.

6. **Maximum number of iterations**: It is another termination criterion. This is the maximum number of iterations when the program is run. The measure collected at this time is whether the criterion was reached, i.e., a solution of a given quality, or not. This value was set at 1,000 iterations.

For convenience, the min-max principle is employed in both examples as the robust design criterion. In order to finally obtain an updated metamodel, ten updating procedures are implemented. It must be pointed out that this work only concerns itself with the performance of MGCPSO. It focuses mainly on such questions as whether or not the optimum of a robust design problem can be determined in this way at all and whether - in terms of computing effort - it has a satisfying parallel efficiency. For this reason, it only takes into account such measurements as final the updated model, its optimum for robust problem, computing time as well as speed-up ratio are taken into account. The performance of the model updating strategy has been sufficiently studied in [70].

4.4.1 Branin Function

The Branin function can be expressed as:

$$f(x,z) = \left(z - \frac{5.1}{4\pi^2}x^2 + \frac{5}{\pi}x - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x) + 10 \tag{4.57}$$

where $x \in [-5, 10]$ is the design variable and $z \in [0, 15]$ is the noise variable. Its original form is shown in figure 4.4(a) and its global robust design is marked with a small diamond. It is usually chosen as test function for global optimization algorithms because it is multi-modal along x axis. 10 initial sampling points are collected by LHD. The final kriging model obtained via MGCPSO is shown in figures 4.4(b) and 4.5(b) . The final kriging model obtained via DIRECT is shown in figures 4.4(c) and 4.5(c) . It can be seen that these two final kriging models look different due to different infilling points during updating procedures. The first difference becomes apparent in the 7th iteration, the infilling point obtained by MGCPSO is $(-0.9336, 15)$, while the infilling point obtained by DIRECT is $(-5, 0)$. The curve of the expected value of improvement $E(I)$ for this step is illustrated in figure 4.7. It can be seen that the global optimum exits in quite narrow a region which may cause the optimization algorithm to miss this tiny region. However, MGCPSO successfully locates the global optimum $x' = -0.9336$ because of its good searching ability. This difference leads to the different final updated model. The real global optima of Branin function for robust design with minimax criterion is 72.3711 at $x = -0.8797$. The robust optima of final obtained model from MGCPSO is 72.4798 at $x = -0.8796$, and in the case of DIRECT, the optima is 72.6826 at $x = -0.8635$. It can be concluded that the result from MGCPSO is better than that from DIRECT. However, both solutions are quite close to the real optimum of the original

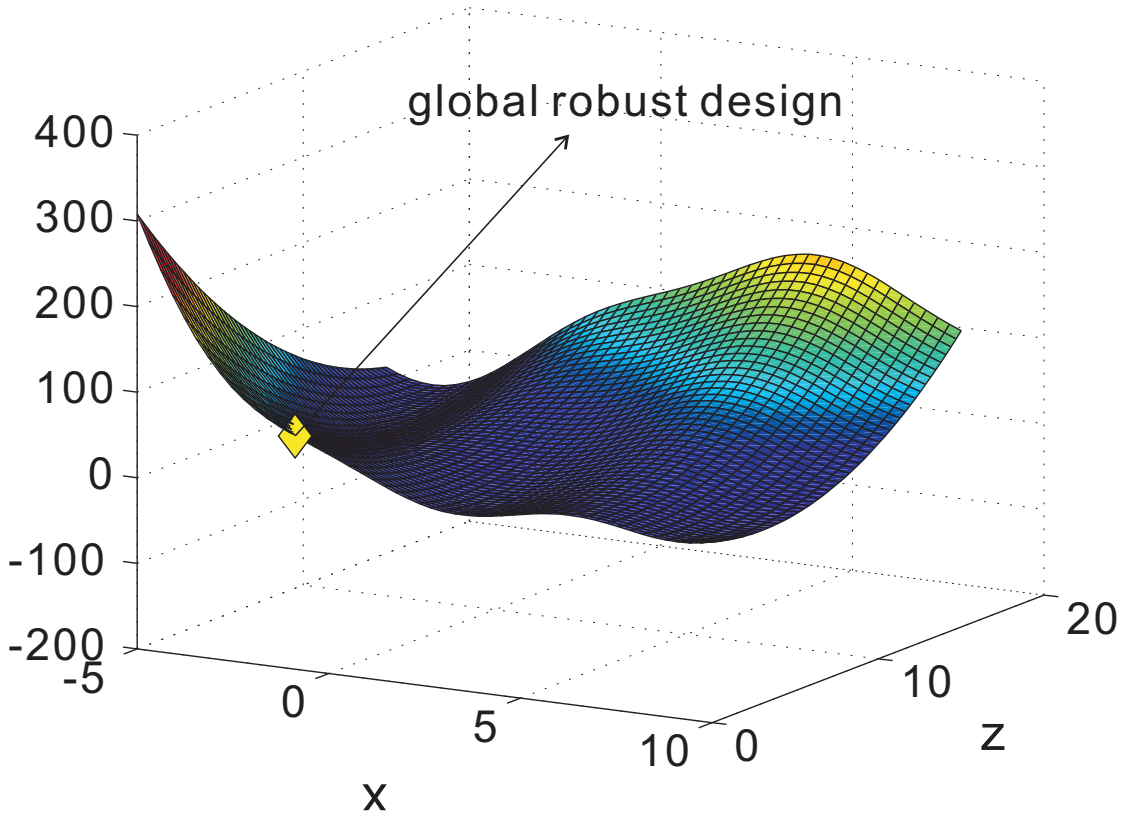
function. The MSE of the final krigingmodle is shown in figure 4.6(a) for MGCPSO and 4.6(b) for DIRECT. From these two figures, it can also be concluded that the MGCPSO performs better that DIRECT in this test because it can find the global optimum for all of the sub-optimization problems and thus the final model can describe the original function very well. Any deviation in the mode updating procedure that is caused by the local convergence of the optimization algorithm can be fixed by inserting more points. Another reason for holding that the model updating strategy is a competent tool in robust design by metamodel is that it does not require the finding of a global optimum for each sub-optimization problem. There does not exist a global optimization algorithm which can be guaranteed to find the global optimum for all kinds of optimization problems. This is proven by the No Free Lunch Theorem discussed in chapter 2. The computing effort and speed-up rate are discussed in 4.4.3.

4.4.2 Camelback Function

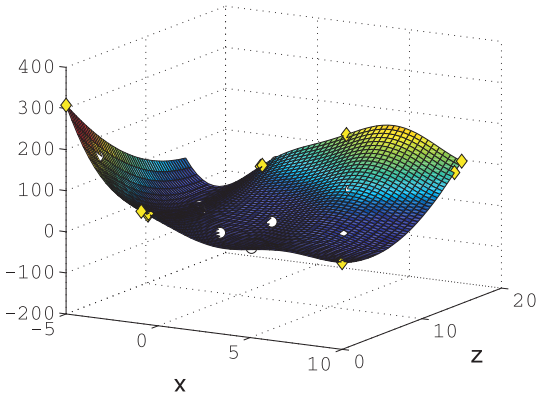
The second numerical example is the six hump camel back function and is written as:

$$f(x,z) = 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xz - 4z^2 + 4z^4 \tag{4.58}$$

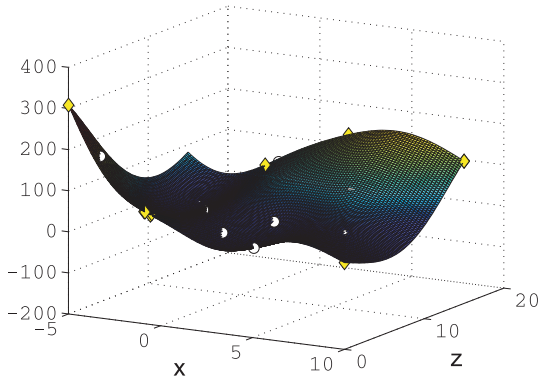
where $x \in [-2,-2]$ is the design variable and z is the noise variable and restricted to the range $-1 \leq z \leq 1$. The original form is shown in figure 4.8(a) and its projection onto the design space is shown in figure 4.9(a) . The robust optimum design is located at $x = 0$ with $f = 0$. Therefore, in this design, three different noise parameter settings are relevant, namely $z = -1$, $z = 0$ and $z = 1$. Thus, any refinement of the metamodel is expected to occur mainly in these three sub-regions during the updating procedure. The twenty initial points are also sampled via LHD. The final kriging model obtained via MGCPSO is shown in figures 4.8(b) and 4.9(b) . The final kriging model obtained via MGCPSO is shown in figures 4.8(c) and 4.9(c) . Similar to the first example, these two final kriging models look slightly different due to different infilling points during the updating procedures. The first difference appears at the fifth step, the figure of $E(I)$ is illustrated in figure 4.11(a) . Note that MGCPSO finds the real global optima at $x = 0.028$ with $E(I) = 0.0171$, while the solution from DIRECT is located at $x = -0.027$ which is not an optimum. The reason why DIRECT finds this position is illustrated in figure 4.11(b) , where for $x = -0.027$ the real global optimized robustness is at $z = -1$ with $f(x,z) = 0.0096$ and DIRECT finds a local minimum at $z = 0$ with $f(x,z) = 0.001655$ and thus leads to a wrong $E(I) = 0.0178$ which is greater than the real global optima $E(I) = 0.0171$. However, the distance between these two different infilling points is not very large so that following infilling points from both algorithms are not affected widely. This causes the final models from these two optimization algorithms to be similar. This conclusion can also be derived from figures of their MSE (seen in figure 4.10(a) for MGCPSO and 4.10(b) for DIRECT). The robust optima of the final updated model from MGCPSO is 0.002876 at $x = 0.002192$ and that from DIRECT is 0.0031 at $x = 0.003015$. Both of them are near the real robust optima, see figure 4.9(b) for MGCPSO and 4.9(c) for DIRECT compared with the original form in figure 4.9(a) . It is concluded that most of the infilling points of both models stay in the expected regions and the robust optimum found in both kriging models (seen in figure 4.8(b) for MGCPSO and



(a) Branin function



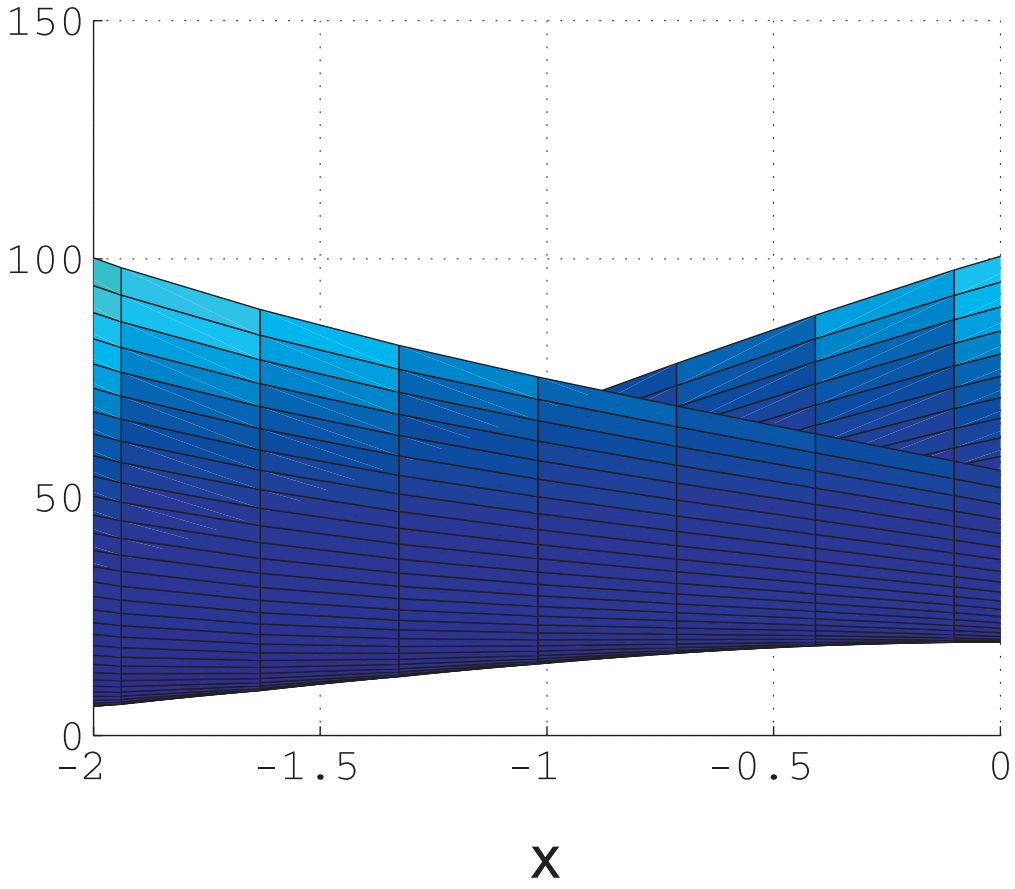
(b) Final kriging model obtained via MGCPSO



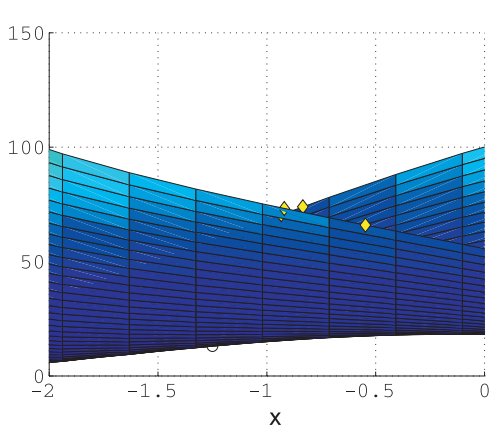
(c) Final kriging model obtained via DIRECT

Figure 4.4: Model of Branin function in isometric view

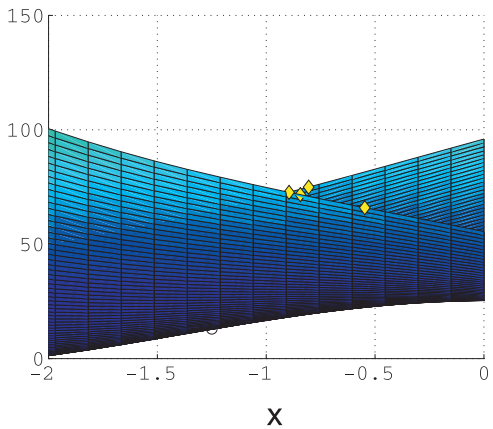
4.8(c) for DIRECT) are very close to the real optimum of the original function, which again confirms that the model updating strategy is very stable and efficient. In the next section, computational efforts are discussed and a conclusion presented.



(a) Projection of original model onto design space (x-y-plane)

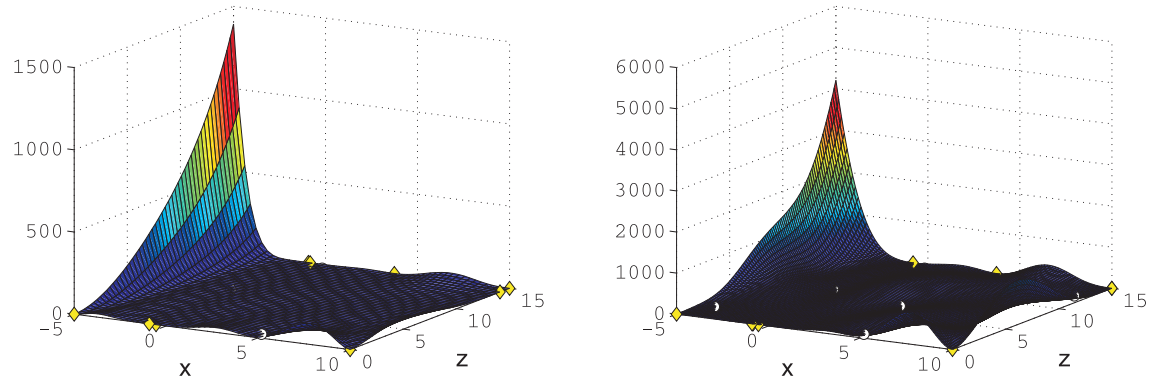


(b) Projection of final kriging model obtained via MGCPSO



(c) Projection of final kriging model obtained via DIRECT

Figure 4.5: Projection of Model of Branin function onto design space (x-y-plane)



(a) MSE of final kriging model obtained via MGCPSO (b) MSE of final kriging model obtained via DIRECT

Figure 4.6: MSE of final kriging model of Branin function

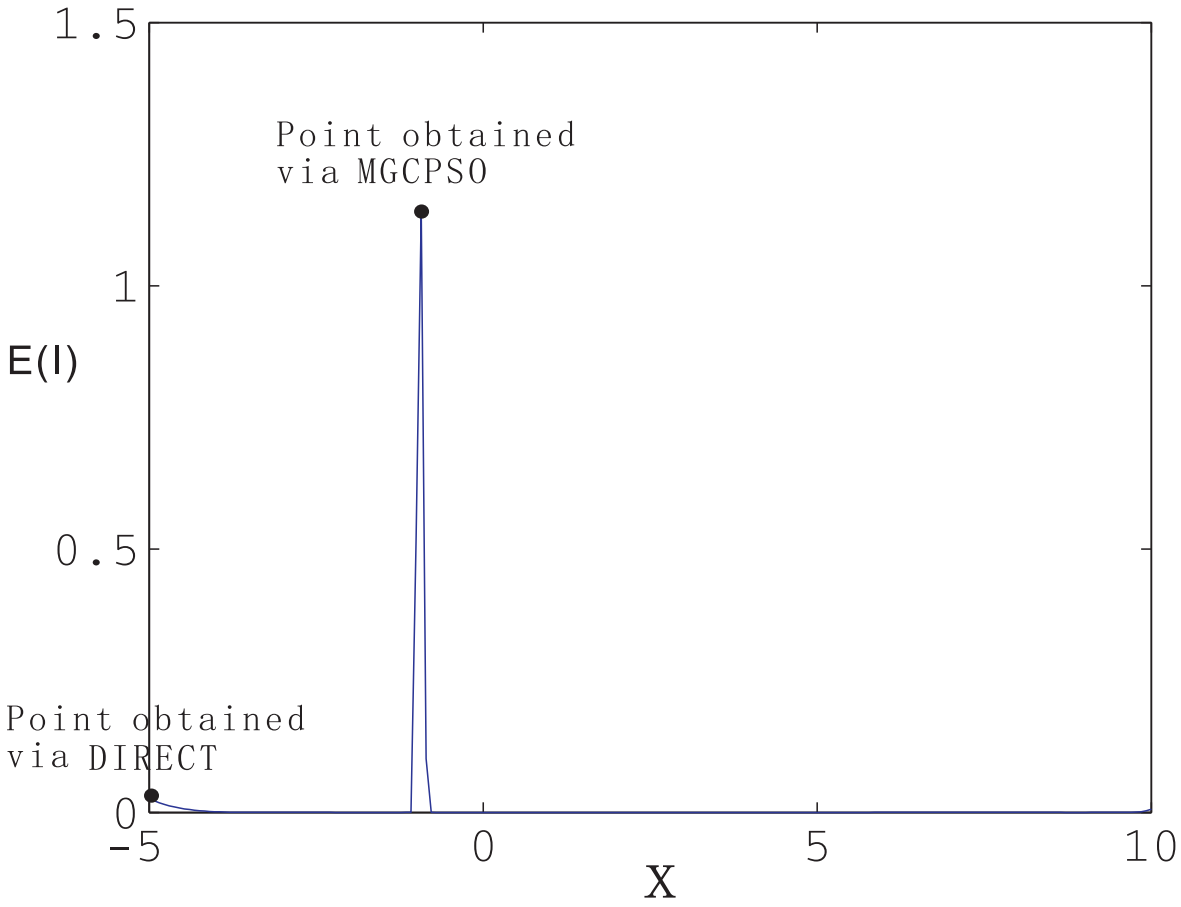
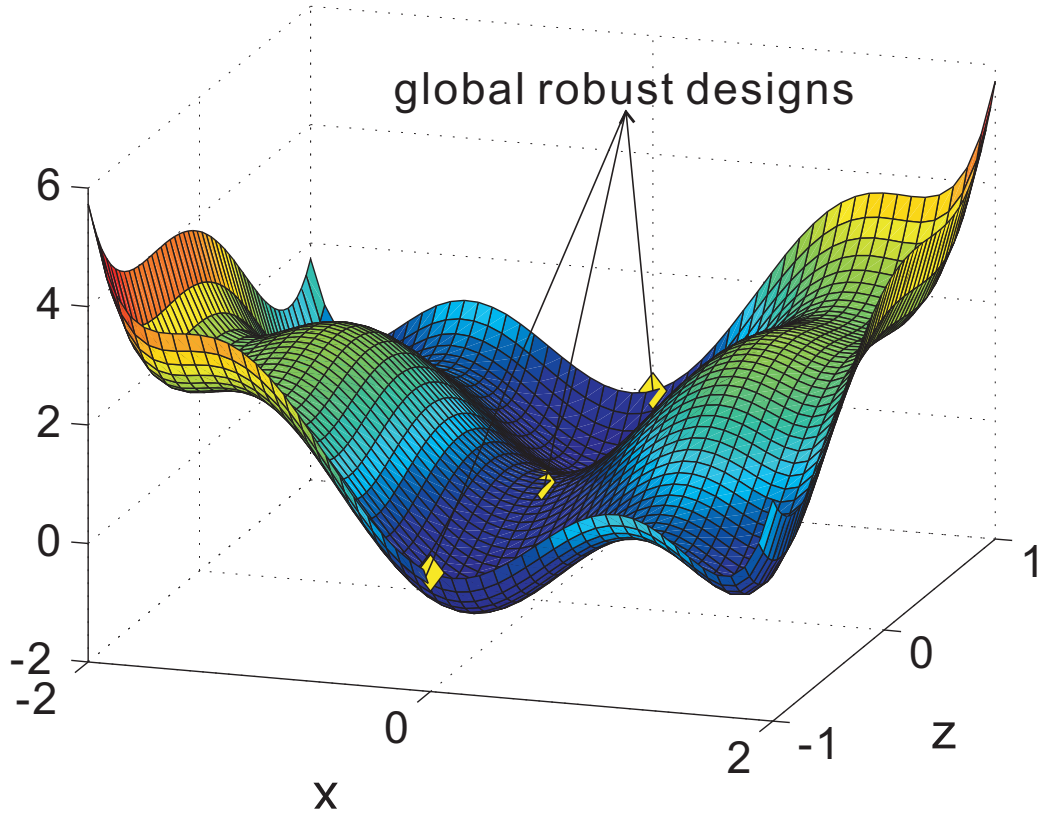
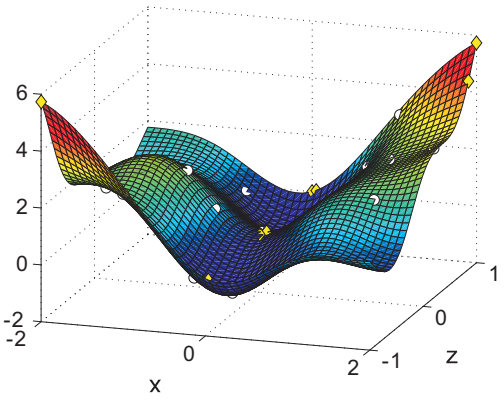


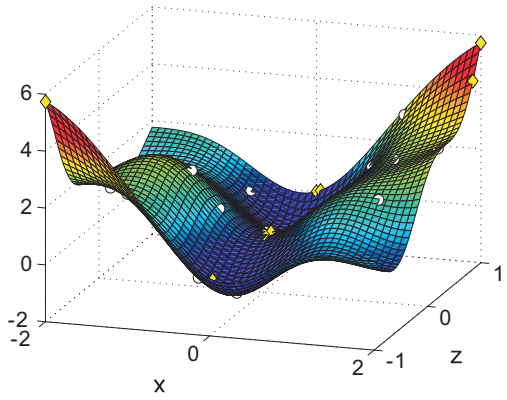
Figure 4.7: $E(I)$ function at seventh step



(a) Camelback function

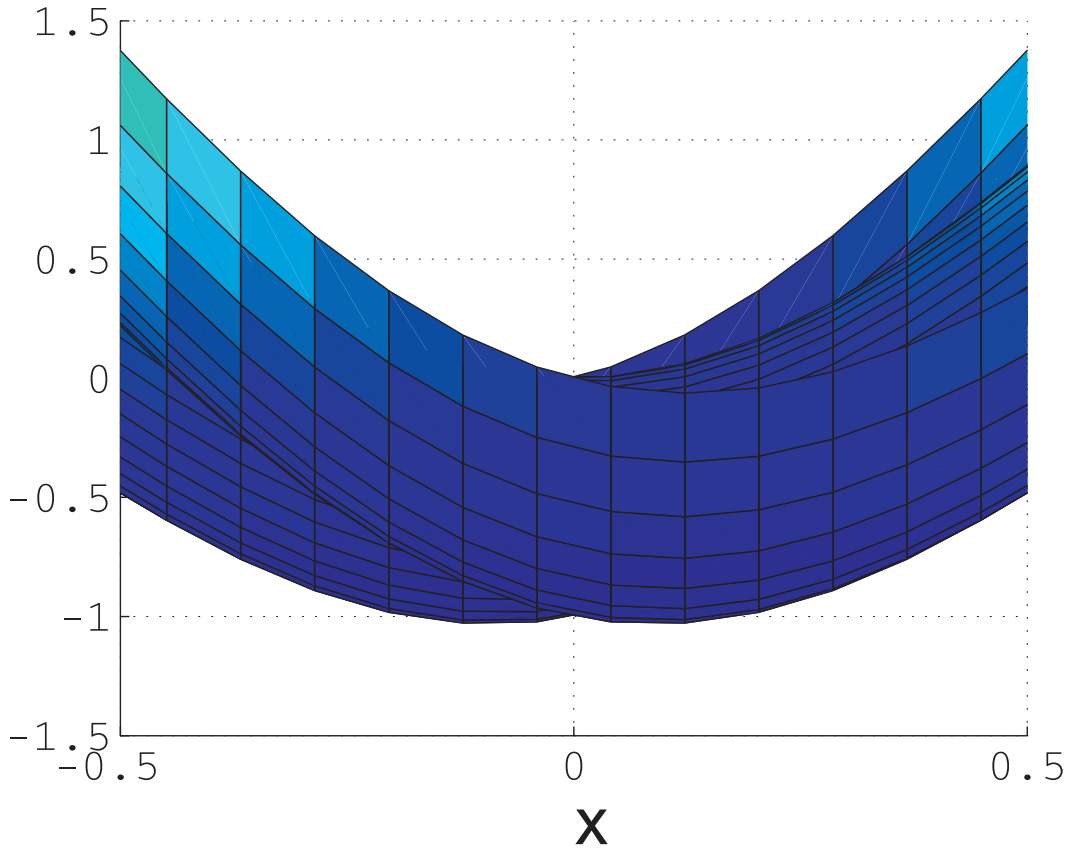


(b) Final kriging model obtained via MGCPSO

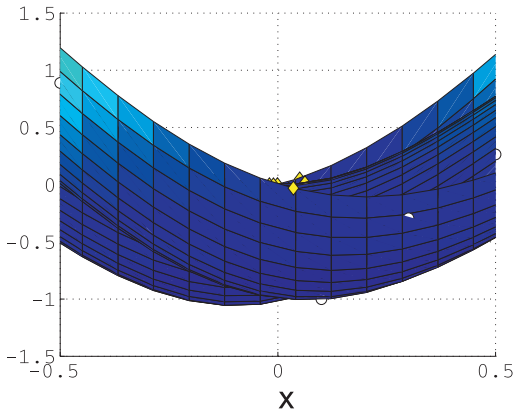


(c) Final kriging model obtained via DIRECT

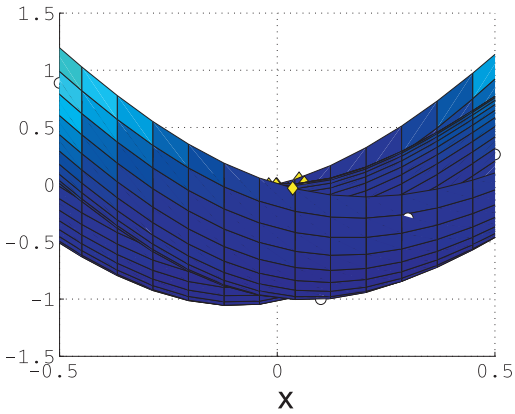
Figure 4.8: Model of Camelback function in isometric view



(a) Projection of original model onto design space (x-y-plane)

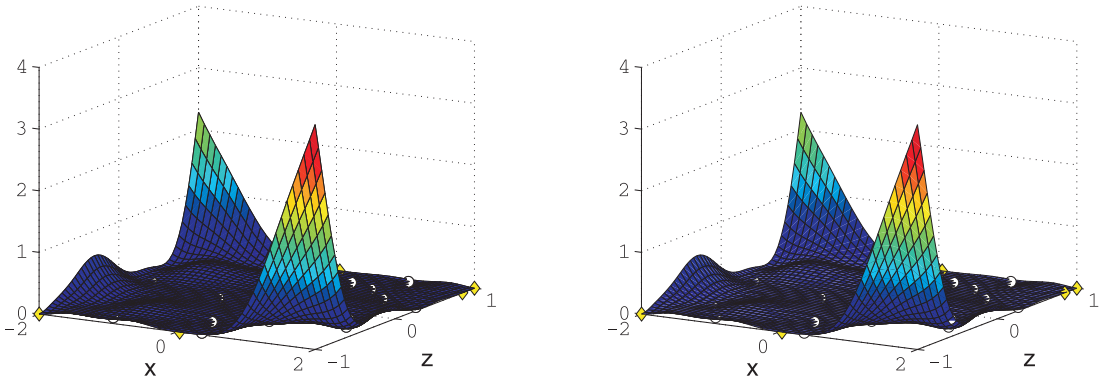


(b) Projection of final kriging model obtained via MGCPSO



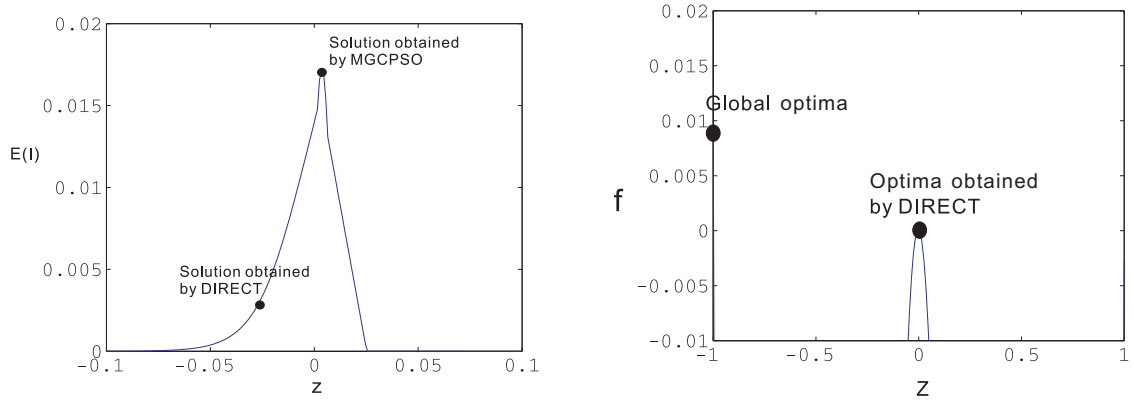
(c) Projection of final kriging model obtained via DIRECT

Figure 4.9: Projection of Model of Camelback function onto design space (x-y-plane)



(a) MSE of final kriging model obtained via MGCPSO (b) MSE of final kriging model obtained via DIRECT

Figure 4.10: MSE of final kriging model of Camelback function



(a) $E(I)$ function at fifth step (b) $f(x,z)$ at $x = -0.0027$

Figure 4.11: The branch of two algorithms

4.4.3 Computational efforts and Conclusion

It has already been mentioned that one of the advantages of parallelized MGCPSO is the reduction of the computing effort. Because ten particles are used for both tests, computing nodes are selected from the sequence $[1, 2, 5, 10]$. Computing time of MGCPSO is listed in table 4.1. From this table, it can be seen that the computing time can obviously be reduced if

	MGCPSO			
	k=1	k=2	k=5	k=10
Branin Function	40s	22s	8.4s	4.4s
Camelback Function	56s	30.2s	11.8s	6s

Table 4.1: Computing time of MGCPSO

parallel computing is used. Moreover, the computing time of the Branin function is less than that of the Camelback function because the number of initial points (10) of the first example is less than that (20) of the second example, i.e. more samplings points will result in a more complex metamodel which increases the time of evaluating the predictions of the points in the design domain by means of metamodel.

The speed-up rate is shown in figure 4.12. Note that the performance of these two examples is quite similar and is not as good as that obtained in numerical tests discussed in chapter 3. The main reason is that the model construction procedure is done only by the master computing node, all the other slave computing nodes have to be waiting still at that time until they receive the model information scattered by the master node. If the model construction procedure can also be done in parallel, the speed-up rate can be further increased. Another reason is probably the efficiency of the interface between Matlab and Fortran. Computing effort may be lost during data exchanging between two different kinds of code.

Finally, it is concluded that:

1. The MGCPSO can obtain competitive results compared with those from DIRECT, robust optima of which are quite close to the real robust optima based on min-max criterion.
2. The MGCPSO exhibits competitive performance in this benchmark test, meaning that it can find global optima in such optimization problems where the DIRECT fails.
3. Using parallel computing can further reduce computing time, which is more obvious if comparing computing effort with that of non-scalable optimization algorithm.
4. The program written in mixed Fortran-Matlab runs very well which dose not acquire additional license cost from Matlab for parallel computing.
5. The model updating strategy is very robust, it can eliminate the deviation caused by local converge of the optimization algorithm in a sub-step by subsequent updating procedures.

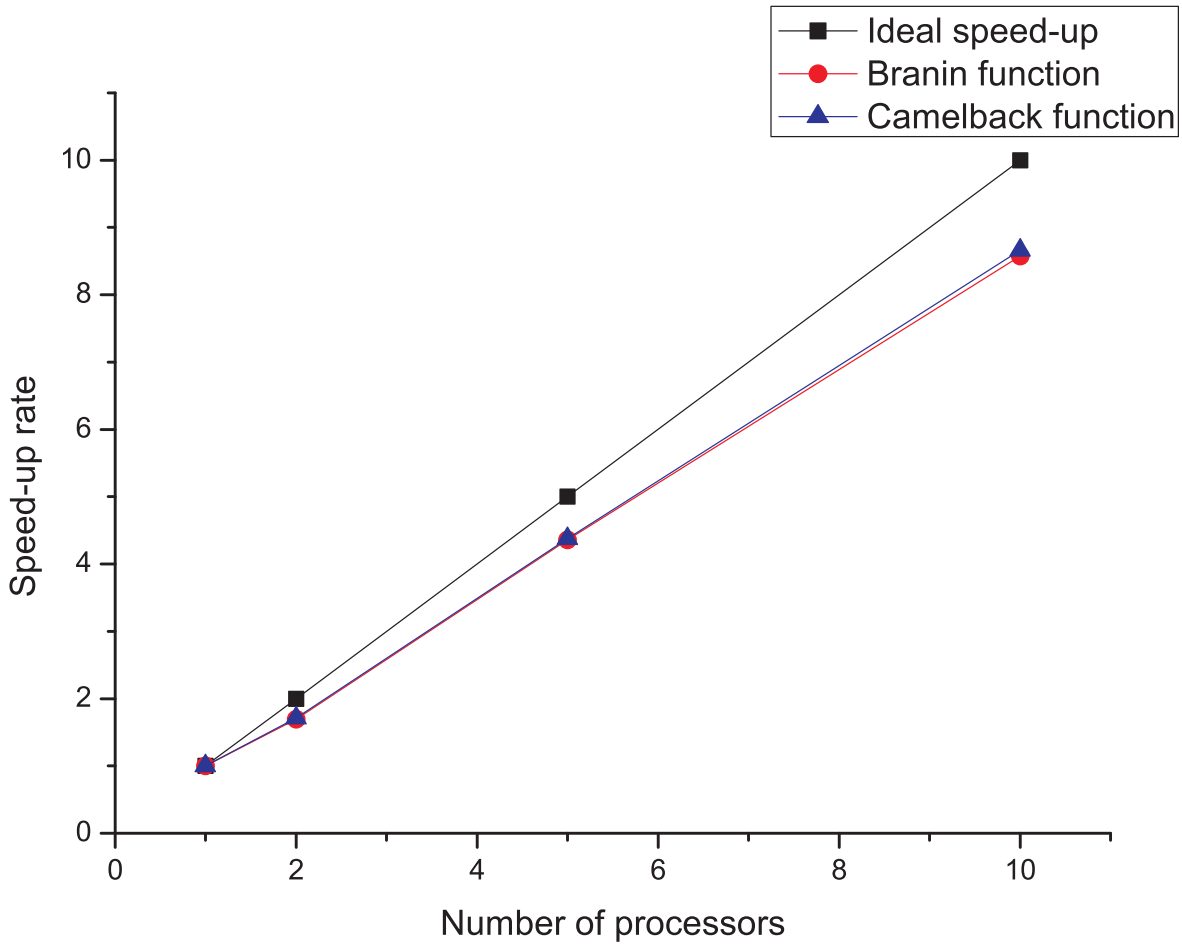


Figure 4.12: Speed-up rate in both numerical examples

Chapter 5

Apply PSO to structural sizing and topological design

In this chapter, conventional structural optimization is first reviewed, such as sizing optimization, shape optimization as well as topological optimization. This is followed by a brief introduction of the common approaches to truss topological optimization. As a new proposed global optimizer, the LPSO is applied to truss topological optimization in the next section. Thereafter, the global search ability of LPSO is tested against a benchmark test alongside examples from actual buildings. In order to expand application field of MGCPSO, it is successfully applied to structural sizing optimization problem. Finally, computational effort and conclusion are presented.

5.1 Conventional structural optimization

Conventional structural optimization is a central branch of optimization, which aims to find a best output that maximises benefit for the designer or decision maker. Until recently, the method has been successfully applied in the automotive, aerospace and civil engineering industries. The rapid development of structural optimizations has been catalyzed by real-life problems, aided by the evolution of sophisticated computing techniques and the extensive applications of the finite element method. As a result, structural optimization now plays an indispensable role in structural design.

A structural optimization problem is usually comprised of five parts:

1. Design variables: The free parameters which need to be determined in order to obtain the expected system performance.

2. Objective functions: These evaluate the merits of the solutions within different sets of design parameters.

3. Design constraints: The constraints imposed by the design of the problem, such as permissible maximal displacement and maximal stress. These are normally retrieved by design codes or experiences.

4. Side constraints: The bounds imposed upon the design variables (such as minimal and maximal values for section areas). Together with the design constraints, they define a feasible domain in the design space.

5. optimization criterion: These consider optimization problems, in light of other constraints, such as maximizing structural stiffness or minimizing structural weights. In the interest of convenience, this thesis is only concerned with determining the minimal of the objective functions, leaving other possible considerations aside.

For a conventional structural optimization problem, the objective functions and design constraints (such like structure compliance, nodal displacements, stresses) may not be expressed as explicit function of design variables. Thus evaluation of these functions demands numerical simulation, such as the finite element method. The structural optimization problem can be expressed mathematically as:

$$\begin{array}{lll} \min\limits_{\mathbf{x}} & f(\mathbf{x}) & \\ \text{subject to} & g_i(\mathbf{x}) \leq 0 \quad ; & i = 1, \cdots, n_g = \mathbb{E} \\ & h_j(\mathbf{x}) = 0 \quad ; & j = 1, \cdots, n_h = \mathbb{I} \\ & x_k^L \leq x_k \leq x_k^U \quad ; & k = 1, \cdots, m \end{array} \tag{5.1}$$

where $\mathbf{x}$ is the vector of design parameters, f is the objective function, $\mathbf{g}$ and $\mathbf{h}$ are inequality and equality constraints respectively and x_k^L and x_k^U are the lower and upper limits of the design variables respectively. Therefore, the feasible domain can be defined as $C = \{\mathbf{x} \in \mathbb{R}^m \mid g_i(\mathbf{x}) \leq 0 \wedge h_j(\mathbf{x}) = 0 \wedge x_k^L \leq x_k \leq x_k^U\}$ and the design space can be characterized by $D = \{\mathbf{x} \in \mathbb{R}^m \mid x_k^L \leq x_k \leq x_k^U\}$.

Here objective functions and design constraints assume their nominal values, which differs from the robust design discussed in chapter 4. In practical applications it is common to use the variable linking technique to reduce the number of independent design variables. This is done by imposing a relationship between the coupled design variables, which is equivalent to an equality constraint in terms of coupled design variables. According to different kinds of design variables, structural optimization can be classified into three different disciplines, which are sizing optimization, shape optimization and topology optimization, respectively, and will be depicted following.

Structural sizing optimization. In this approach, the design variables $\mathbf{x}$ are some type of structural thickness, e.g., cross-sectional areas of truss members, the thickness distribution of a sheet, or beam section parameters. A classic application of sizing is the determination of the minimal cross-sectional areas in truss structures which is illustrated in figure 5.1. It is a two-bar truss problem and can be stated as:

$$\begin{array}{lll} \min\limits_{D} & & f(D) = 2\rho\pi DT(B^2 + H^2)^{1/2} \\ \text{s.t.} & \text{stress constraints} & \dfrac{P(B^2 + D^2)^{1/2}}{\pi TDH} \leq [\sigma] \\ & \text{stability constraints} & \dfrac{P(B^2 + D^2)^{1/2}}{\pi TDH} \leq \dfrac{\pi^2 E(D^2 + T^2)}{8(B^2 + H^2)} \end{array} \tag{5.2}$$

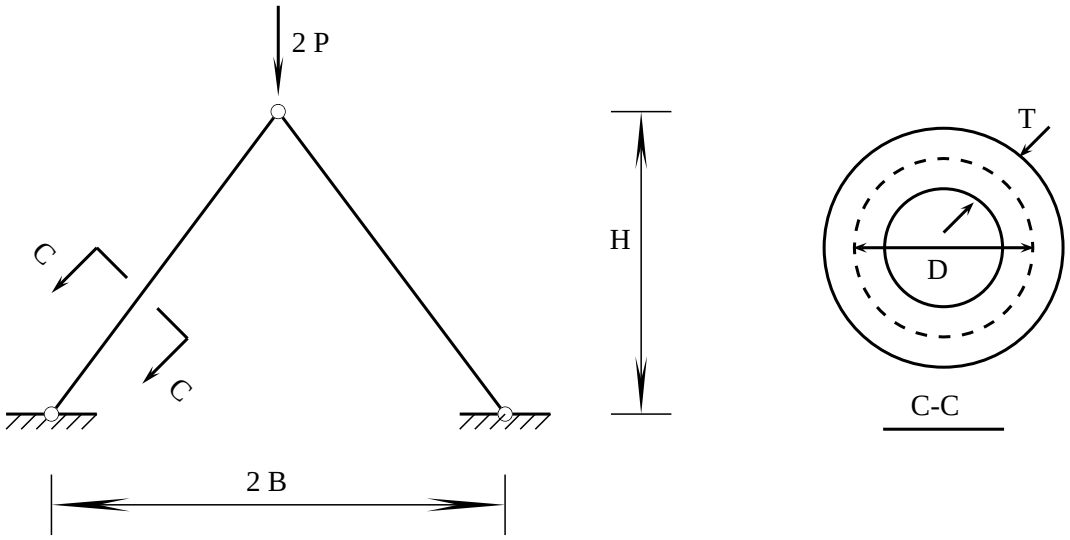


Figure 5.1: A typical sizing structural optimization problem

where the design variable is cross section diameter D, other parameters are: the applied central load $2P = 600\,\text{KN}$, the horizontal distance $2B = 6\,\text{m}$ between two supports, Young's modulus $E = 2.1 \times 10^5\,\text{MPa}$, materiel density $\rho = 78\,\text{KN/m}^3$, allowable stress $[\sigma] = 160\,\text{MPa}$, structure height $H = 4\,\text{m}$. Since it is a simple problem, a conventional optimization method can be used for finding the optimum $f(D^*) = 1.8256\,\text{KN}$ with $D^* = 0.149\,\text{m}^3$.

Structural shape optimization. In shape optimization, structural geometry parameters (such as the position of joints or the position and shape of an internal cavity) are defined as design variables and used to optimize system performance. Normally, shape optimization problems can be treated similar to sizing optimization, the main difference is to use different kinds of design variables. For demonstrating shape optimization, the classic two-bar truss shown in figure 5.1 is still used. However, in this case, the height H of the structure is adopted as design parameter and the diameter D of section area is still taken into account. It shall be noted that the dash lines in figure 5.2 represent the possible designs during optimization procedure. This optimization problem can be stated as:

$$
\begin{aligned}
&\min_{H,D} && f(H,D) = 2\rho\pi DT(B^2+H^2)^{1/2} \\
&\text{s.t.} \quad \text{stress constraints} && \frac{P(B^2+D^2)^{1/2}}{\pi TDH} \le [\sigma] \\
&\qquad \text{stability constraints} && \frac{P(B^2+D^2)^{1/2}}{\pi TDH} \le \frac{\pi^2 E(D^2+T^2)}{8(B^2+H^2)} \\
&&& 2m \le H \le 6m
\end{aligned}
\tag{5.3}
$$

where the design variables are H and D and other design parameters are the same as for the sizing optimization problem. Because this example merely serves the purpose of demonstration, the optimum can easily be obtained, i.e. $f(H^*, D^*) = 1.769\,\text{KN}$ with $H^* = 3\,\text{m}$

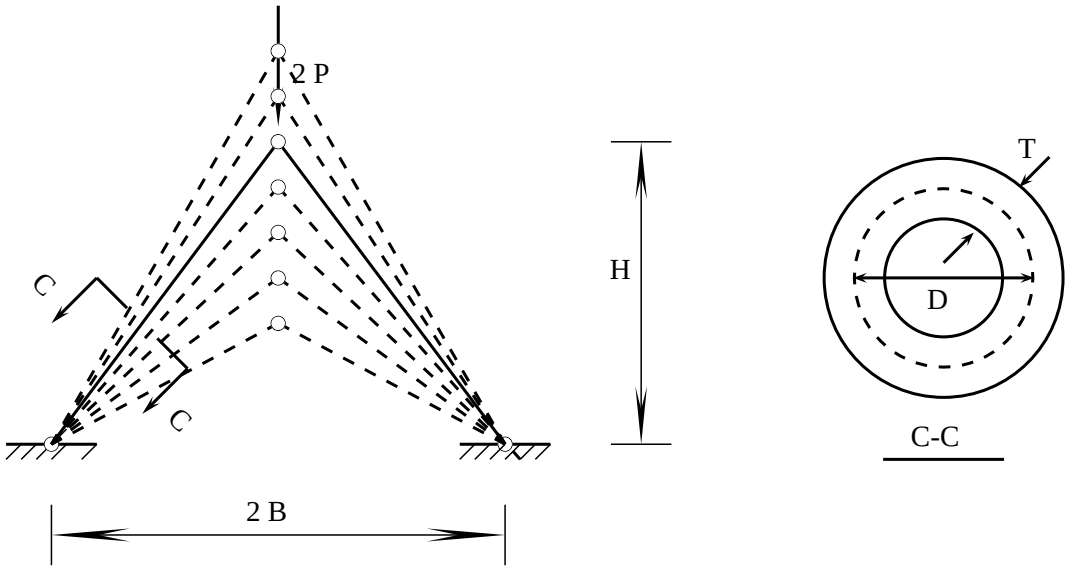


Figure 5.2: A simple structural shape optimization problem

and $D^* = 0.169\,\text{m}$. Compared with the optima from the last example, it is noted that it is possible to further reduce the structural weight by using optimization geometry.

Structural topology optimization This is the most general form of structural optimization and requires a less detailed description of the concept than the other two kinds of optimization problem. The design variables in topology optimization describe the structural configuration. Topology optimization is a difficult problem and it has received more attention in applications to skeletal structures such as trusses and frames. In these cases, the optimum criterion can be defined by determining which joints are connected to each other by members. The basic and most widely used approach is to initially create a ground structure where every joint is connected to other joints. Members' cross-sectional areas are taken as design variables, and these variables are allowed to take the value zero during optimization procedure, i.e. members are removed from the truss. Finally, a concise and optimized structural layout is obtained. This approach was first proposed by Dorn et el. in [37]. A very simple example is shown in figure 5.3.

In the initial stage of structural design (also called conceptual design), it is often desired to find a general layout that can naturally and efficiently resist the anticipated design loads. This is sometimes done by optimizing the overall shape as well as the topology of the structural system. Ideally, shape optimization can be thought of as a subclass of topology optimization, however practical implementations are based on quite distinct techniques. Thus, these two types are normally regarded as separate. From a fundamental point of view, topology and sizing optimization are very different matters. However, from a pragmatic stand point similarities between the two categories are apparent. When the stated problem is a differential equation, it can be said that shape optimization concerns the controlling of the equation's domain, whilst sizing and topology optimization concern the control of its parameters [103]. The research work in this chapter focuses on structural sizing and topology optimization.

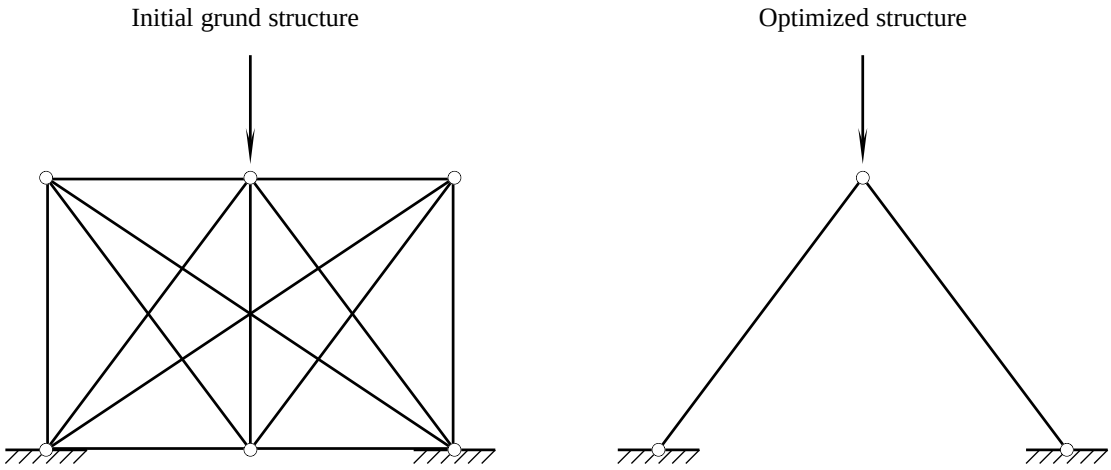


Figure 5.3: A demonstration of structural topology optimization

5.2 Apply LPSO to truss topology optimization

Techniques for optimizing the topology of a structural system can be classed into the following two categories:

1. Discrete optimization
2. Continuum optimization

In discrete optimization, the structure is generally modeled with discrete truss, beam or column elements, whereas in continuum optimization the structure is modeled as a continuum. There has been extensive research on both kinds of optimization techniques in the past two decades, and both have their own strengths and weaknesses. Since this work only focuses on discrete topological optimization, truss topological optimization is discussed in this and the next sections. However, frames (rigidly-jointed structures) can be also by employing the finite element method and they are easier to handle than trusses (pin-jointed structures) due to the stability at the joints.

5.2.1 An overview of truss topology optimization

The initial study of the fundamental properties of optimal grid like continua was pioneered by Michell [90] which was important in view of the theoretical background. However, the numerical methods in this field have a shorter history which appeared following the initial developments of high-speed computers. Early contributions can be found in Dorn, Gomory and Greenberg [37] and Fleron [49] in which numerical implementations were first proposed and exercised on very small test problems due to computing limitations. Since the 1980's, there has been an unprecedented and most dramatic growth in computing technologies. Since then, the theoretical work on structural topology optimization has continued to unfold. To illustrate this, Rozvany [111] obtained new optimality conditions (Continuum-like Optimization Criterion (COC)) of the Michell truss which lead to different

and lighter trusses compared with those from Michell. Rozvany [112] pointed out shortcoming in Michell's truss theory, Zhou [143] listed difficulties in truss topology optimization with stress and local buckling constraints, exact solutions of some truss layout optimization were derived. In particular, Lewiński, Zhou and Rozvany [79] derived exact least-weight truss layouts for rectangular domains with various support conditions, as well as exact analytical solutions for some popular benchmark problems in topology optimization from Rozvany [113], Lewiński and Rozvany [77, 78]. On the other hand, numerical approaches have been developed and applied to larger-scale, more realistic structures. Two fundamental techniques were proposed for this kind of optimization problem: evolution and degeneration. The evolution approach is a growing and heuristic approach, in which the basic structure is a simple bar truss and finial optimal topology is generated by adding nodes and members[114]. Although the use of this approach can avoid unrealistic or unstable optimal solutions, there is no theoretical criterion for addition of nodes and members. On the contrary, as a representative of the degeneration approach, the ground structure technique was first introduced by Dorn et el. [37] and is now widely used in all kinds of truss topological optimization problems. In this approach, the nodal locations are fixed and the ground structure is created by connecting any two nodes. During the optimization procedure, unnecessary members are removed. Many methods have been presented based on the ground structure approach. Two normal kinds of ground structure are shown in figure 5.4. On the left side of figure 5.4, the member length is restricted to a certain number which expresses the fact that spectrum of possible member lengths can be restricted and can thus be viewed as a reduced form. As a result, the computational effort is also reduced. However, the optimal topology may not be the global optimum because some connecting members are ignored which may belong to the optimal candidates. The ground structure that can be seen on the right side is known as a fully connected ground structure and owns the set of all possible connections between every two chosen nodal points. This approach consumes, of course, more computer resources, producing, in turn, more exact solutions. Although the ground structure approach can now be regarded as a standard procedure in the field of truss topological optimization, there are indeed several difficulties which can be summarized in the following way [97]:

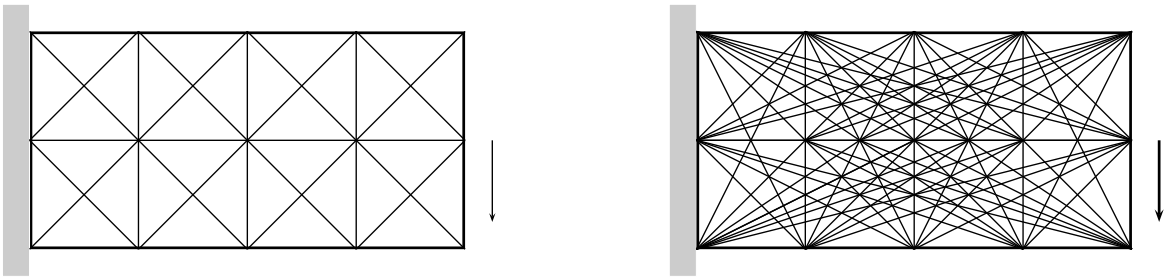
1. A large amount of members and nodes are needed in the initial ground structure. The number of non-overlapping members can be calculated by:

$$Sum(x,y) = 1/2xy(xy-1) - \sum_{i=1}^{x}\sum_{j=1}^{y} f(i,j) \tag{5.4}$$

where

$$f(i,j) = \begin{cases} \{i=1 \wedge j \geq 3\} \vee \{i \geq 3 \wedge j = 1\} & (x-i+1)(y-j+1) \\ (i-1:j-1) \geq 2 & 2(x-i+1)(y-j+1) \\ \text{else} & 0 \end{cases} \tag{5.5}$$

where x and y are the horizontal and vertical number of grids respectively, $(i:j)$ is an operator to compute the maximal common divisor between i and j.



(a) (b)

Figure 5.4: Two kinds of classic truss ground structures for transmitting a vertical force to a vertical line supports

2. The optimal topology strongly depends on the initial design and infinite number of nodes and members are needed if the nodal locations are also to be optimized(such as simultaneous shape and topology optimization).

3. Unrealistic optimal solutions are often obtained.

4. The truss may lose stability if too many members are removed.

5.2.2 Problem formulation of truss topology optimization and its equivalences

In this work, the simplest possible optimal design problem (P1), namely the minimization of compliance (maximization of stiffness) for a given total mass of the structure, is considered. Several classic problems of this kind can be seen as a standard benchmark test for optimization algorithms due to its high-dimensional and non-convex features. The well-known formulation of problem P1 is expressed as:

$$\begin{aligned} \text{P1} \quad \min_{\mathbf{x}} \quad & \mathbf{f}^T\mathbf{u} \\ \text{subject to} \quad & \sum_{i=1}^{m} x_i \mathsf{K}_i \mathbf{u} = \mathbf{f} \\ & \sum_{i=1}^{m} x_i = V \\ & x_i \geq 0, \quad i = 1, \cdots, m \end{aligned} \tag{5.6}$$

where x_i is the volume of the ith bar and $x_i\mathsf{K}_i$ is the element stiffness matrix for the ith bar written in global coordinates. If all of the member cross-section volumes are considered to be greater than zero, then P1 is transformed to a truss sizing optimization problem. Therefore the stiffness matrix $\mathsf{K}(\mathbf{x}) = \sum_{i=1}^{m} x_i\mathsf{K}_i$ is positive definite for all eligible $\mathbf{x}$ and the displacements can be removed from the problem. As a consequence, the resulting problem with only bar volume design variables is proved to be convex and solutions are confirmed [126]. If zero lower bound is allowed, i.e. the bars of the ground structure can be removed, then the problem statement covers topology design. Furthermore, the stiffness matrix is no longer guaranteed to be positive definite and the vector $\mathbf{u}$ can not be removed by solving $\mathsf{K}\mathbf{u} = \mathbf{f}$, if the zero lower bound is taken into account. Nevertheless, it is not important to remove

$\mathbf{u}$ from formula (5.6), while, typically, the number m of the bars is much greater than the number of degrees of freedom. It can be seen that in the complete ground structure, the total amount of all possible members can be calculated as $m = n(n-1)/2$, while the degrees of freedom are only $2n$ for planar trusses or $3n$ for 3-d trusses. As a consequence, several approaches focus on reducing this problem to a displacement only problem through equivalent formulations of problem 5.6 which will be discussed later.

If Lagrange multipliers $\hat{\mathbf{u}}$, Λ, $\mu_i, i = 1, \cdots, m$ are induced, problem (5.6) can be stated as:

$$L = \mathbf{f}^T\mathbf{u} - \hat{\mathbf{u}}^T(\sum_{i=1}^{m} x_i\mathsf{K}_i\mathbf{u} - \mathbf{f}) + \Lambda(\sum_{i=1}^{m} x_i - V) + \sum_{i=1}^{m}\mu_i(-x_i) \tag{5.7}$$

The necessary conditions can be obtained by differentiating formula (5.7)

$$\sum_{i=1}^{m} x_i\mathsf{K}_i\hat{\mathbf{u}} = \mathbf{f}, \quad \hat{\mathbf{u}}^T\mathsf{K}_i\mathbf{u} = \Lambda - \mu_i, \quad \mu_i \leq 0, \quad \mu_i x_i = 0, \quad i = 1, \cdots, m.$$

Now a new symbol $\Lambda^*(\mathbf{u})$ is defined to denote the maximum of the individual bar's specific energy $\mathbf{u}^T\mathsf{K}_i\mathbf{u}$ and is given by

$$\Lambda^*(\mathbf{u}) = \max_{i=1,\cdots,m}\left\{\mathbf{u}^T\mathsf{K}_i\mathbf{u}\right\}$$

$J(\mathbf{u})$ is used to denote the set of bars which hold the maximum specific energies and are namely active bars

$$J(\mathbf{u}) = \left\{i \mid \mathbf{u}^T\mathsf{K}_i\mathbf{u} = \Lambda^*(\mathbf{u})\right\}$$

Then the necessary conditions are satisfied with

$$\begin{aligned} \hat{\mathbf{u}} = \mathbf{u}; \quad x_i = \hat{x}_i V, i \in J(\mathbf{u}); \quad x_i = 0, i \notin J(\mathbf{u}); \quad \Lambda = \Lambda^*(\mathbf{u}); \\ \mu_i = 0, i \in J(\mathbf{u}); \quad \mu_i = \Lambda^*(\mathbf{u}) - \mathbf{u}^T\mathsf{K}_i\mathbf{u}, i \notin J(\mathbf{u}); \\ V\sum_{i\in J(\mathbf{u})}\hat{x}_i\mathsf{K}_i\mathbf{u} = \mathbf{f} \end{aligned} \tag{5.8}$$

where $\hat{x}_i = x_i/V$ is the non-dimensional element volume. It can be proved that there does indeed exist a pair $(\mathbf{u}, \mathbf{x})$ which satisfies the necessary condition (5.8) and it automatically constitutes a minimizer for the non-convex form (5.6). A detailed proof can be found in [128].

Note that the equilibrium condition of problem (5.6) can be expressed in terms of the principle of minimum potential energy. Therefore, displacement $\mathbf{u}$ is the minimizer of the structural total potential energy $F(\mathbf{v}) = \frac{1}{2}\mathbf{v}^T\left(\sum_{i=1}^{m} x_i\mathsf{K}_i\right)\mathbf{v} - \mathbf{f}^T\mathbf{v}$. Note that the value $\mathbf{F}(\mathbf{u})$ is equal to $-\frac{1}{2}\mathbf{f}^T\mathbf{u} < 0$. Therefore, the problem (5.6) can be rewritten as a max-min problem in this form

$$\max_{\substack{x_i\geq 0, i=1,\cdots,m \\ \sum_{i=1}^{m} x_i = V}} \min_{\mathbf{u}}\left\{\frac{1}{2}\mathbf{u}^T\left(\sum_{i=1}^{m} x_i\mathsf{K}_i\right)\mathbf{u} - \mathbf{f}^T\mathbf{u}\right\} \tag{5.9}$$

This is a saddle point problem for a concave-convex problem and the max and min operator can be interchanged. Thus formula (5.9) can be stated as

$$\min_{\mathbf{u}} \max_{\substack{x_i \geq 0, i=1,\cdots,m \\ \sum_{i=1}^{m} x_i = V}} \left\{ \frac{1}{2} \sum_{i=1}^{m} x_i \mathbf{u}^T \mathsf{K}_i \mathbf{u} - \mathbf{f}^T \mathbf{u} \right\} \tag{5.10}$$

The inner problem is a linear programming problem in the $\mathbf{x}$ variable. A very useful inequality is stated as:

$$\sum_{i=1}^{m} x_i \mathbf{u}^T \mathsf{K}_i \mathbf{u} \leq V \max_{i=1,\cdots,m} \left\{ \mathbf{u}^T \mathsf{K}_i \mathbf{u} \right\} \tag{5.11}$$

The equality holds if all the non-zero bars have the maximum specific energy which partly is a solution of necessary condition for problem (5.6). Thus, problem (5.10) can be further reduced to

$$\min_{\mathbf{u}} \max_{i=1,\cdots,m} \left\{ \frac{V}{2} \mathbf{u}^T \mathsf{K}_i \mathbf{u} - \mathbf{f}^T \mathbf{u} \right\} \tag{5.12}$$

This is an unconstrained, convex and non-smooth problem in displacement $\mathbf{u}$ only and its optimal is minus one half of the optimal value of problem (5.6) [15]. This displacement only equivalence has been studied in [2], [15] and [18].

Problem (5.12) can also be written as an equivalent smooth, constrained and convex problem

$$\begin{array}{ll} \min\limits_{\mathbf{u}} & \tau \\ \text{subject to} & \tau - \frac{V}{2} \mathbf{u}^T \mathsf{K}_i \mathbf{u} + \mathbf{f}^T \mathbf{u} \geq 0, \quad i = 1, \cdots, m \end{array} \tag{5.13}$$

Although this problem has a large number m of constraints, it can be solved quickly through interior point methods, e.g. Penalty/Barrier Multiplier Method (PBM) proposed by Bental and Zibulevsky [17]. Relevant research can also be found in [16] and [68].

Achtzinger and Stolpe also proposed two equivalences to problem (5.6). The first equivalence is given by

$$\begin{array}{lll} \min\limits_{\mathbf{u},\mathbf{r},\Lambda} & -\frac{1}{2} \sum\limits_{i=1}^{m} L_i \mathbf{u}^T \mathsf{K}_i \mathbf{u} - V\lambda + \mathbf{f}^T \mathbf{u} + \sum\limits_{i=1}^{m} (U_I - L_i) r_i & \\ \text{subject to} & \frac{1}{2} \mathbf{u}^T \mathsf{K}_i \mathbf{u} - \lambda + r_i \leq 0, & i = 1, \cdots, m \\ & r_i \leq 0, & i = 1, \cdots, m \\ & \lambda \geq 0 & \end{array} \tag{5.14}$$

where L_i and U_i denote lower and upper bounds on the bar volumes, respectively, rather than on the bar areas, l_i is length of the ith bar. Note that L_i and U_i satisfy following inequalities

$$\begin{aligned} 0 \leq L_i \leq U_i < +\infty \qquad & j = 1, \cdots, m \\ \sum_{i=1}^{m} L_i < V < \sum_{i=1}^{m} U_i & \end{aligned}$$

Note that it has a quadratic objective function and quadratic inequality constraints and is therefore referred to as an "all-quadratic optimization problem", sometimes abbreviated as a "Q2P" problem. The numerical solution of (Q2P) can be solved by any large-scale nonlinear optimization program. Let $(\mathbf{u}, \mathbf{r}, \lambda)$ be the optimal for problem (5.14) with corresponding Lagrange multipliers $\tau_j, \quad j = 1, \cdots, m$ for the quadratic constraints, the $(\mathbf{x}, \mathbf{u})$ is optimal for problem (5.6) where

$$x_i = L_i + \tau_i, \quad i + 1, \cdots, m \tag{5.15}$$

Also, this equivalence is proper for the use of an interior point methods, such as PBM.

The second equivalence can be stated as:

$$\begin{array}{lll} \min\limits_{\mathbf{u},\mathbf{s},\mu} & -\frac{1}{2}(\mathbf{u}^T, \mathbf{s}^T, \mu)C(\mathbf{u}^T, \mathbf{s}^T, \mu)^T + \mathbf{f}^T\mathbf{u} & \\ \text{subject to} & \mathbf{b}_i^T\mathbf{u} - s_i - \mu \leq 0 & i = 1, \cdots, m \\ & -\mathbf{b}_i^T\mathbf{u} - s_i - \mu \leq 0 & i = 1, \cdots, m \\ & s_i \leq 0, & i = 1, \cdots, m \\ & \mu \geq 0 & \end{array} \tag{5.16}$$

where $\mathbf{b}_i = \dfrac{\sqrt{E_i}}{l_i}\gamma_i$ by γ_i is a vector of direction cosines (note that (scaled) stiffness matrix K_i can be written as a dyadic product $\mathsf{K}_i = \mathbf{b}_i\mathbf{b}_i^T, \quad i = 1, \cdots, m$) and E_i is the Young's modulus of the material used for the ith bar, and matrix C is given by

$$C = \begin{pmatrix} \mathsf{K}_L & 0 & 0 \\ 0 & diag(\mathbf{U} - \mathbf{L}) & \mathbf{U} - \mathbf{L} \\ 0 & (\mathbf{U} - \mathbf{L})^T & V_L \end{pmatrix}$$

where

$$\mathsf{K}_L = \sum_{i=i}^{m} L_i\mathsf{K}_i, \quad i = 1, \cdots, m$$

Note that although this is not a convex QP optimization problem (i.e. its Hessians is indefinite) and most QP-solvers require the Hessian of the objective function to be positive definite or, at least to be positive semidefinite, it can be solved efficiently by computing its KKT-point $(\mathbf{u}, \mathbf{s}, \mu)$ which can be proved to be the global optimizer of (5.16). This is because the numerical calculation of KKT-points of this non-convex problem is possible in an fast way by making use of the QP problem structure. Another stimulating feature for this equivalence is that the vectors $\mathbf{L}$ and $\mathbf{U}$ only appear in the Hessian C of the objective function. Therefore, the constraint set of QP remains unchanged for different volume bounds. Let $(\mathbf{u}, \mathbf{s}, \lambda)$ be the optimal of problem (5.16) with corresponding vectors δ^-, δ^+ of multipliers for the first two sets of inequality constraints, then $(\mathbf{x}, \mathbf{u})$ is an optimal of problem (5.6) where

$$x_i = \begin{cases} L_i + \dfrac{1}{\mu}[((\delta_i^- - \delta_i^+) - (U_i - L_i)s_i)] & if \quad U_i > L_i \\ L_i & if \quad U_i = L_i \end{cases} \qquad i = 1, \cdots, m \tag{5.17}$$

These two equivalences are included in the Branch-and-Bound method for computing the lower bound of the integer optimization problem stated in 5.26. Details about how to

apply Branch-and-Bound method to solve integer truss topology optimization problem can be found in [5], [4] and [3].

As natural extensions of problem (5.6), to find the optimal topology of the reinforcement of a given structure and the optimal topology problem with self-weight can be taken into account, thus formula (5.6) can be expressed as

$$\begin{aligned} \min_{\mathbf{x},\mathbf{s}} \quad & \left[\mathbf{f} + \sum_{i\in R} x_i \mathbf{g}_i + \sum_{i\in \mathbb{S}} s_i \mathbf{g}_i\right]^T \mathbf{u} \\ \text{subject to} \quad & \left[\sum_{i\in \mathbb{R}} x_i \mathsf{K}_i + \sum_{i\in \mathbb{S}} s_i \mathsf{K}_i\right] \mathbf{u} = \mathbf{f} + \sum_{i\in \mathbb{R}} x_i \mathbf{g}_i + \sum_{i\in \mathbb{S}} s_i \mathbf{g}_i \\ & \sum_{i=1}^{m} x_i = V \\ & x_i \geq 0, \quad i = 1, \cdots, m \end{aligned} \tag{5.18}$$

where $x_i, i \in \mathbb{R}$ is the bar volumes and $s_i, i \in \mathbb{S}$ is the reinforcement part, $\mathbf{g}_i$ denotes the specific nodal gravity vector due to the self-weight of the ith bar. This problem can be solved in analogous ways which is demonstrated in the solving of formula (5.6). It can also be written in similar equivalent forms.

As a relatively recently developed global optimizer, PSO has been successfully applied to structural sizing optimization problems in the past decades. However, there were fewer applications to truss topology optimization. The main aim of this thesis is to use modified particle swarm optimizers to solve truss topology optimization. Questions of truss topology optimization that show minimal compliance with a given volume, as stated in (5.6), can efficiently be solved by employing equivalent formulations. However, these equivalences are all based on the optimality criteria which is derived from the necessary condition (5.8). As soon as a new objective function arises and/ or new constraints are added, the original equivalence looses its validity. The acquisition of a new equivalence requires a strong mathematical background (most researchers who work on truss topology optimization and equivalences in particular come from institutes of mathematics). For this reason, engineers with an interest in learning more about solving truss topology design problems find access to the field quite challenging. EAs have been applied to the field of truss topology optimization, albeit in a limited manner since their application is limited to problems of a small scale. Unfortunately, they are thus not suited for real projects. Consequently, it is necessary to develop an algorithm that is easy to implement (such as a particle swarm optimizer) for solving large-scale problems. Since truss topology optimization is non-convex, highly multi-modal and the standard PSO is proved to be a local convergent algorithm, a modified PSO is required for handling any difficulties associated with truss topological problems. Two modified PSOs are proposed in chapter 3. Their performances are also evaluated via a benchmark test, from which it is noted that the LPSO can solve high multi-modal problems very well and the MGCPSO owns a trade-off performance between results and computational efforts compared with standard PSO and LPSO. This section discusses the application of LPSO, MGCPSO, and their respective parallel patterns to the subject of truss topology optimization with real, as well as integer design variables. Specifics are dealt with in the following subsections, as are the numerical experiments.

5.2.3 Geometry consistent check

This approach is based on the ground structure where a large number of potential nodes and an even larger number of potential bars are distributed over a design domain, so that the mathematical optimization problem is formulated in terms of real/integer cross-section areas of the bars to design variables and the displacements to state variables. Two common kinds of ground structure are demonstrated in figure 5.4. Since partially disconnected ground structures can lead to designs that are not ideal for the chosen set of nodal points - in other words, they are local minimum - this work only considers fully connected ones. One of the difficulties inherent in fully connected ground structures is their high flexibility, meaning that their connecting members can be added or removed freely during the optimization procedure. In order to avoid the computing of unrealistic structures, and thereby reducing the computing effort, it is essential to perform a consistency check with regard to geometry before structural analysis. This is the case because PSO constitutes a global stochastic search algorithm and the intermediate structure may be a mechanism or have redundant members (It may cause loss of parallel performance, which is discussed later). Several common potential cases that need to undergo a geometry consistency check are shown in figure 5.5. A very important assumption is that all trusses are elastic structures and can thus be analyzed by means of the linear elastic finite element analysis. It must be noted that the substructures shown in the left column in figure 5.5 are taken from the overall structure. The strategy followed can be described in the following terms:

1. In case (a), node a and node c are connected through bar 1 and bar 2 with an inner node b. Because all bar members are suppressed by axial load, the inner node b can be eliminated and bar 1 and bar 2 can be merged into bar 3 with the volume $t_3 = t_1 + t_2$.

2. In case (b), node a is connected by bar 1 and bar 2 and not suppressed by an external load. Based on the elastic theory, the stresses of bar 1 and bar 2 are zero, so that bar 1 and bar 2 can be removed from the structure and node a is eliminated also.

3. In case (c), similar to case (a), bar 3 can be seen as a free member, i.e. if there is no external load and/or displacement constraints on node d, bar 3 will have a rigid motion. Therefore, node d and bar 3 need to be removed from the structure, bar 1 and bar 2 are combine into bar 4 and node b is consequently eliminated, as in case (a).

4. In case (d), an external load is applied to an isolated node, i.e. the external load cannot be transfered to the structure's boundary. Thus, this case is, of course, inapplicable to problem (5.6), so that it can be ignored with regard to further. Note, that it may cause an unbalanced task in parallel computing, since the computing node, in this case, will not analyse the structure but output a predefined large value and then stand in an idle status, meanwhile other computing nodes are still executing structural analysis.

5. In case (e), the resulting structure is in equilibrium under the given load. In another direction, however, namely vertical to the load direction, the structure is a mechanism. Consequently, the resulting structure is not applicable in reality which is common in ground structure approach.Although this particular difficulty can be overcome by the addition of bars to the structure to make it stable, some questions remain unanswered:

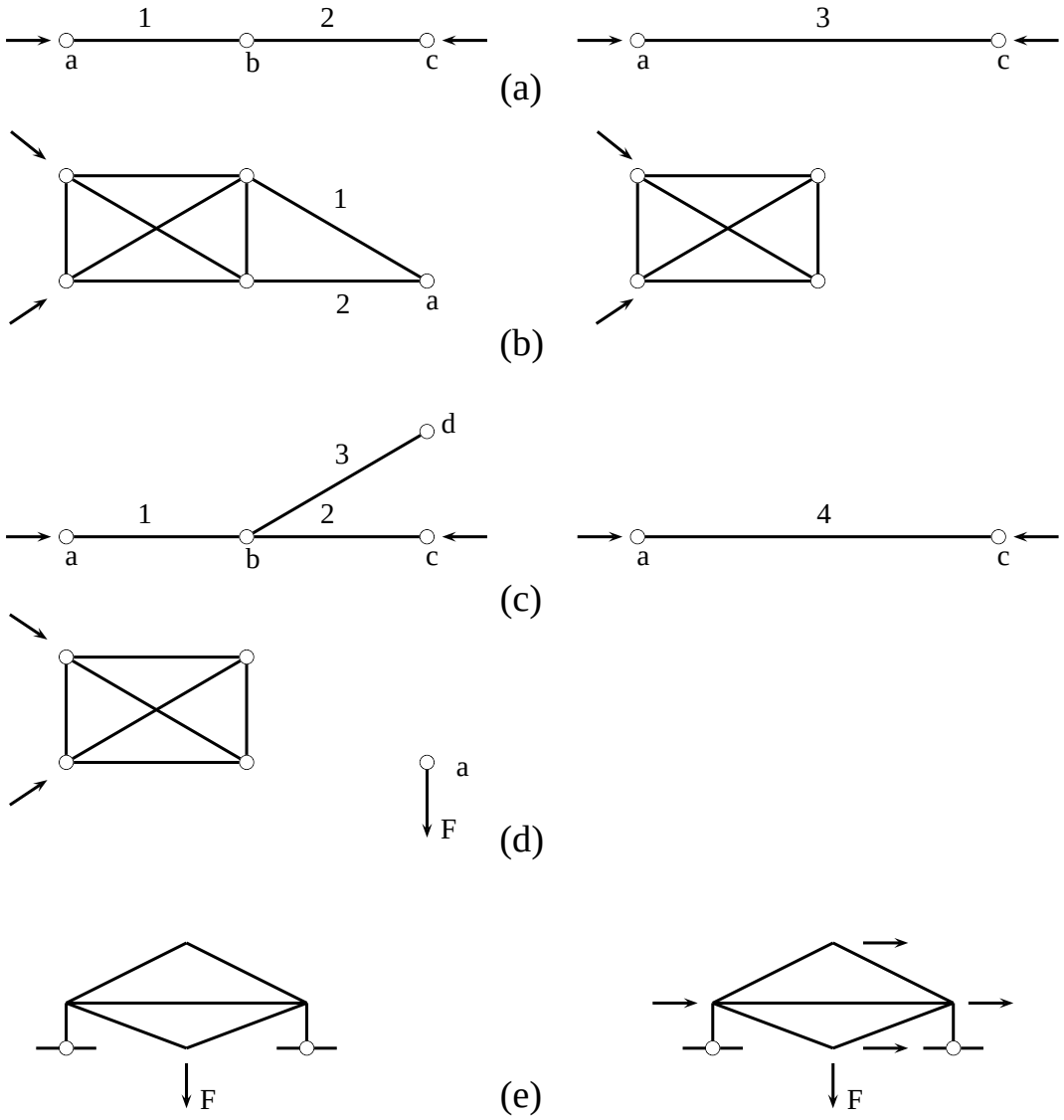


Figure 5.5: Geometry consistent check

How can bars be added heuristically? How can the volumes of the additional bars be determined and the same amount of volume is subtracted from the existing members so that the volume constraint still holds? This problem can be resolved by inserting minor external loads to each of the nodes. The added loads are vertical to the existing external loads. If the nodal displacements in that direction are not larger than a certain pre-defined value then it can be concluded that this structure is not a mechanism any more. Admittedly, this strategy lacks theoretical backing but the fact remains that it is an efficient method and that can be shown in the numerical experiments below.

Case (a)highlights that it is impossible for overlapping members to appear simultaneously with sub-members. As a result, the dimension of the design variables can be reduced from the number $1/2N(N-1)$ (where N is the number of all the nodes) of fully possible members to the number of non-overlapping members which can be computed via formula (5.5).

5.2.4 How to handle constraints

The goal of optimization problems with constraints is to find a solution that optimizes the fitness function whilst also satisfying the given constraints. The search area in constrained optimization problems is divided into feasible and infeasible domains. Within the feasible domain all points satisfy all constraints, whereas inside the infeasible domain all points violate at least one of the constraints. Because PSO and evolutionary algorithms have many properties in common, it is possible to use the same method for treating constraints in both cases. Carlos Artemio Coello Coello [30] has summarized in a survey all of the theoretical and numerical techniques that are appropriate for handling constraints. So far the main categories of the constraint-handling techniques are:

1. Penalty functions, e.g. external penalty, internal penalty, death penalty.
2. Special representations and operators, e.g. Random keys.
3. Repair algorithms.
4. Separation of objectives and constraints, e.g. co-evolution, behavioral memory.
5. Hybrid methods, e.g. Lagrange multipliers, fuzzy logic.

Because of the No-Free-Lunch Theorem [136] it is known that it is impossible to create a universal constraint-handling technique which is able to treat all kinds of constraints with most excellent performance. The penalty function technique is a compromising and conservative way of dealing with constraints. Also, it allows for easy implementation and offers effective solutions to most types of optimization problems that have been tested.

A common approach of penalty function technique is called sequential unconstrained minimization technique (SUMT) that was first proposed by Fiacco and McCormick [46]. SUMT transforms a given constrained optimization problem into a sequence of unconstrained optimization problems. This transformation is accomplished by defining an appropriate auxiliary function, in terms of problem function, to define a new objective function whose optima are unconstrained in some domain of interest [47]. Thus, in the case of SUMT, a sequence of penalty functions is defined where the penalty terms for the constraint violations are multiplied by some positive coefficient, so that the constrained optimization problems are transformed into a sequence of unconstrained but penalized optimization problems which can be solved by all kinds of optimization methods. By penalizing constraint violations more and more severely, the minimizer is forced to the feasible region for the constrained problem. There are two main variants of the penalty technique:the internal and the external penalty function. The most poignant difference between them is that the external penalty function searches only the infeasible domain while the internal penalty function searches only the feasible domain. Often, minima of external penalty functions are inapplicable to the original problem and the solution only grows feasible when the penalty parameter grows extreme large. Consequently, the generated solutions may not always be useful for real-life applications. On the contrary, in the case of internal penalty functions, the penalty term acts as a barrier to prevent the searching points from leaving the feasible

region. As a result, minima of the internal penalty function are always feasible. For this reason, it lends itself to inequality constraints. The disadvantage of this approach is that it can only handle strict inequality constraints.

Note that in problem (5.6) there exist equality and inequality constraints. Inequality constraints are applied on section areas of bars which can be handled directly by setting proper intervals for design variables by the boundary-check of PSO. Thus, these inequality constraints are automatically guaranteed to be fulfilled. Equality constraints can be dealt with by employing a exterior quadratic penalty function. For stochastic algorithms, equality constraints can only rarely be satisfied since in the case of equality constraints the feasible domain is reduced to quite a narrow region and the particles search the design space randomly. As a consequence, the exterior penalty function is used to handle constraints of this sort in this work.

The exterior quadratic penalty function for equality constraints is given by

$$F(\mathbf{x}, \mu) = f(\mathbf{x}) + \frac{1}{2\mu} \sum_{i \in \mathbb{E}} h_i^2(\mathbf{x}) \tag{5.19}$$

where μ is the penalty parameter and $h_i(\mathbf{x}) = 0, \quad i = 1, \cdots, n_h$ are equality constraints. By driving μ to zero, the constraint violations are penalized with increasing severity. It makes good intuitive sense to consider a sequence of value $\{\mu(t)\}$ with $\mu(t) \to 0$ as $t \to \infty$, so that the task is to seek the approximate minimizer $\mathbf{x}^t$ of $F(\mathbf{x}, \mu(t))$ for each t. It is noted that PSO is not affected by the ill-conditioned properties of (5.19). The convergence of the quadratic penalty function can be proved in the following way. It is supposed that $\mathbf{x}^t$ is the global minimizer of $F(\mathbf{x}, \mu(t))$ and $\hat{\mathbf{x}}$ is the global solution of original optimization problem. Since $\mathbf{x}^t$ minimizes $F(\mathbf{x}, \mu(t))$ for each t, we have such inequalities $F(\mathbf{x}^t, \mu(t)) \leq F(\hat{\mathbf{x}}, \mu(t))$, which can be rewritten as:

$$f(\mathbf{x}^t) + \frac{1}{2\mu(t)} \sum_{i \in \mathbb{E}} h_i^2(\mathbf{x}^t) \leq f(\hat{\mathbf{x}}) + \frac{1}{2\mu(t)} \sum_{i \in \mathbb{E}} h_i^2(\hat{\mathbf{x}}) = f(\hat{\mathbf{x}}) \tag{5.20}$$

By rearranging this expression, we obtain

$$\sum_{i \in \mathbb{E}} h_i^2(\mathbf{x}^t) \leq 2\mu(t)[f(\hat{\mathbf{x}}) - f(\mathbf{x}^t)] \tag{5.21}$$

Suppose that $\mathbf{x}^*$ is the limit point of $\{\mathbf{x}^t\}$ and given by

$$\lim_{t \in \mathbb{T}} \mathbf{x}^t \to \mathbf{x}^* \tag{5.22}$$

where K is an infinite subsequence. Thus the following formula can be obtained via formula (5.21) by taking the limit as $t \to \infty, t \in \mathbb{T}$

$$\sum_{i \in \mathbb{E}} h_i^2(\mathbf{x}^*) = \lim_{t \in \mathbb{T}} \sum_{i \in \mathbb{E}} h_i^2(\mathbf{x}^t) \leq \lim_{t \in \mathbb{T}} 2\mu(t)[f(\hat{\mathbf{x}}) - f(\mathbf{x}^t)] = 0 \tag{5.23}$$

where the last equality holds as $\mu(t)$ tends to zero. Therefore, we have that $h_i(\mathbf{x}^*) = 0$ for all $i \in \mathbb{E}$, so that $\mathbf{x}^*$ is feasible. Moreover, due to the non-negativity of $\mu(t)$ and each $h_i^2(\mathbf{x}^t)$, we have that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}^*) + \lim_{t \in \mathsf{T}} \frac{1}{2\mu(t)} \sum_{i \in \mathbb{E}} h_i^2(\mathbf{x}^t) \leq f(\hat{\mathbf{x}}) \tag{5.24}$$

since $\mathbf{x}^*$ is feasible point whose objective value is no larger than that of the global minimizer $\hat{\mathbf{x}}$, it can be concluded that $\mathbf{x}^*$, too, is a global minimizer. More detailed proof can be found in [96]. If inequality constraints are taken into account, a mixed internal-external quadratic penalty approach can be used, thus formula (5.19) can be rewritten as

$$F(\mathbf{x}, \mu) = f(\mathbf{x}) + \frac{1}{2\mu} \sum_{i \in \mathbb{E}} h_i^2(\mathbf{x}) + \mu \sum_{j \in \mathbb{I}} \frac{1}{g_j^2(\mathbf{x})} \tag{5.25}$$

where $g_j(\mathbf{x})$ are inequality constraints.

5.2.5 Integer programming

Truss topology optimization with real design variables has been well studied, however, for the case of topology design with integer section areas, it is more practical if the truss must be built from pre-produced bars with given section areas. This kind of optimization problem can be stated as

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{f}^T \mathbf{u} \\ \text{subject to} \quad & \sum_{i=1}^{m} x_i \mathsf{K}_i \mathbf{u} = \mathbf{f} \\ & \sum_{i=1}^{m} x_i = V \\ & x_i \in \{x_i^1, \cdots, x_i^{m_i}\}, \quad i = 1, \cdots, m \end{aligned} \tag{5.26}$$

where $\{x_i^1, \cdots, x_i^{m_i}\}$ represents all the discrete candidates for ith bar. It belongs to integer programming. Research work in this field can be divided into two classes. One class is based on a deterministic optimization algorithm with rounding strategies, meaning that it solves the problem through an approximation with continuous variables. References to this approach can be found in Ringertz [110], Achtziger and Stolpe [5, 3, 4], where truss problems are handled through brand-and-bounch method. There are further references in Beekers and Fleury [14] where a primal-dual method is used as well as in Stolpe and Svanberg [124] where it is proved that a wide range of problems in 0-1 topology design can be written in mixed integer LP format. Another class is based on stochastic optimization algorithm, successful approaches can be found in Hajela and Lee [58] (genetic algorithm), in Topping et el. [129] (simulated annealing). Note that Achtziger and Stolpe have successfully applied the brand-and-bounch approach to large-scale numerical examples and obtained guaranteed global optimum. It is important to remember that their approach relies heavily on the use of efficient equivalent formulations as discussed in 5.2.2. Consequently, a deep understanding of this method supposes a strong mathematical background. However, evolutionary algorithms (EAs) can be applied directly to optimization problems that are in the original form. In case the acquisition of new constraints is necessary, the only task is to add them to the penalized objective function with proper penalty functions.

Since PSO was initially only developed for problems defined over real vector spaces, new strategies have to be proposed for a successful extension of it to cases of discrete optimization. Common strategies that allow for the extended application of PSO include binary PSO and the rounding strategy which has already been discussed in subsection (2.5.5). In

this work, the rounding strategy is used for dealing with the integer design variables for MGCPSO and LPSO.

Since PSO is a time consuming algorithm, it is necessary to adopt the parallel pattern discussed in chapter 3 to reduce computing time. Truss topology optimization problems can be managed in a way that is analogous to the benchmark tests discussed in chapter 3. Moreover, before calculating fitness values with the corresponding penalty term (5.25) certain additional procedures should be carried out. To begin with, a fully connected ground structure needs to be constructed at the beginning of the program. Secondly, a geometry consistency check is necessary for as long as the design variables $\mathbf{x}$ are computed. Thirdly, because the state variables $\mathbf{u}$ are not directly supplied, they need to be obtained by means of structural analysis (per finite element analysis, for example) with given $\mathbf{x}$. It is worth noting that all three of these procedures can be executed by the same computing node where the corresponding $\mathbf{x}$ is stored and updated. As mentioned in the foregoing, because a penalty function is employed in handling structural constraints PSO needs to be implemented consequentially in order to obtain a series of $(\mathbf{x}^t)$ by optimizing a set of $F(\mathbf{x}, \mu(t))$ provided that t is the current generation number of penalty factor. The stopping criterion for the whole loop is given by

$$\begin{aligned} &\text{if } \mid F(\mathbf{x}^t, \mu(t)) - F(\mathbf{x}^{t-1}, \mu(t-1)) \mid \leq \epsilon, \text{ then stop algorithm;} \\ &\text{else, update } \mu(t+1) \text{ and optimize new } F(\mathbf{x}, \mu(t+1)) \end{aligned} \tag{5.27}$$

where $\epsilon = 1.0D^{-7}$ is a predefined small value. Finally, the entire work flow is illustrated in brief in figure 5.6.

5.3 Numerical experiments

The parameter set for PSO is analogous to that used in the numerical benchmark tests in chapter 3 and is given by:

1. **Inertia Weight** ω: This coefficient is used to control the trajectories of particles and is set to $\omega = 0.5 + rand()$.
2. **Acceleration Coefficients** φ_1 and φ_2: These are mostly used in the community of particle swarms, the values are $\varphi_1 = \varphi_2 = 1.49445$.
3. **Maximum Velocity** v_{max}: This parameter was not set, as it was not deemed necessary.
4. **Population Size**: All the swarms used in this study comprise twenty individuals.
5. **Stopping criterion**: If the best position of swarm can not be improved in fifty consecutive iterations, the program will be stopped artificially and the fitness of the best position will be considered the result of this numerical test.
6. **Maximum number of iterations**: This is another termination criterion. It constitutes the permissible maximum number of iterations during which the program is run. The measure collected at this time is whether the criterion was reached, i.e., a solution of a given quality, or not. This value was set at 5,000 iterations.

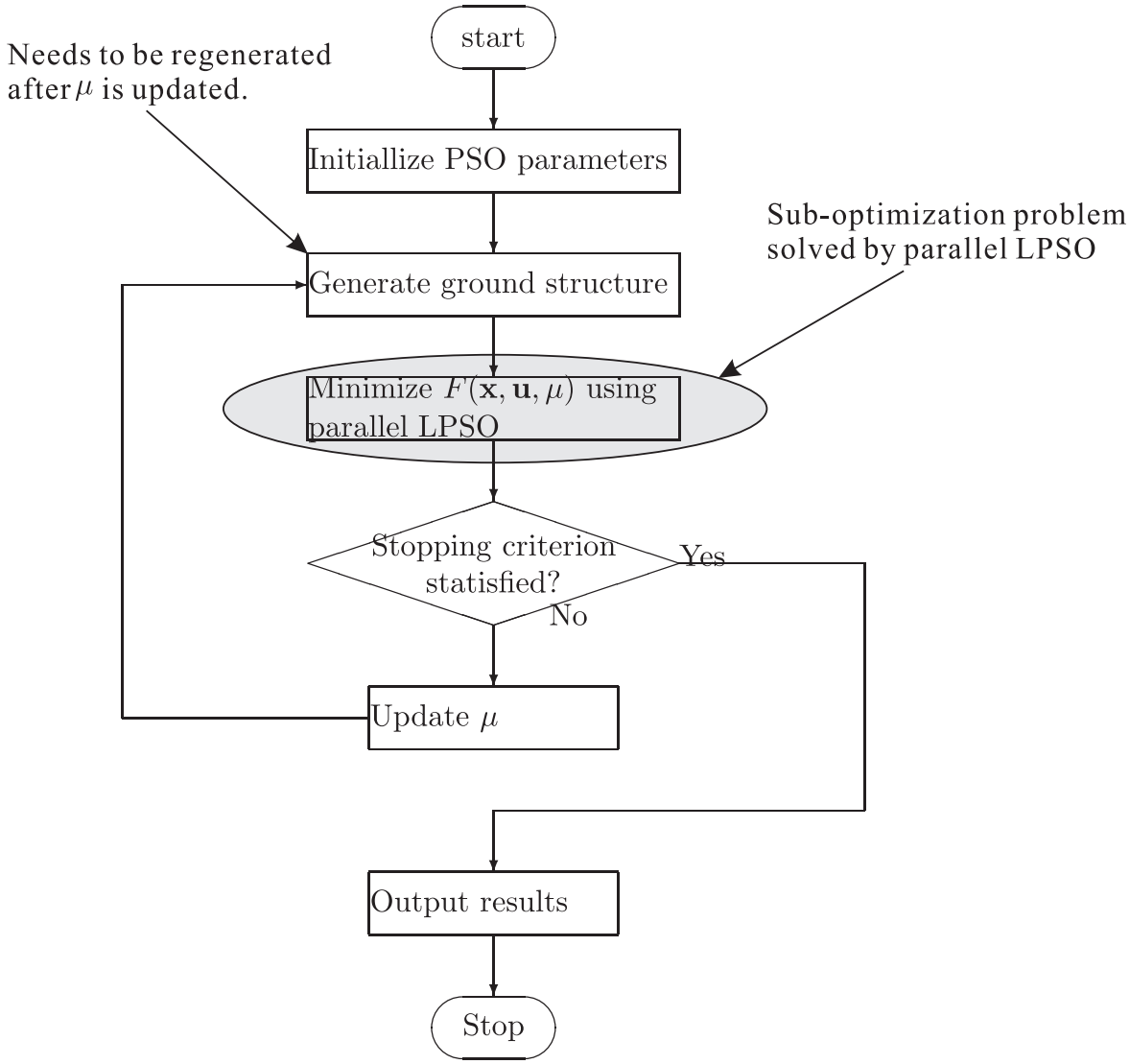


Figure 5.6: Workflow of applying LPSO to truss topology optimization

Additionally, the penalty parameter is updated by $\mu(t) = 10^{5-t}, t \in \{1, 2, ..., 10\}$. In order to study the algorithm's performance, each example is solved using all the possible μ consequentially. Because MGCPSO fails in most of the optimization problems that this benchmark test concerns - meaning that it failed to find a realistic solution, rendering no more successful than the best result achieved so far - its performance needs to be improved. The result obtained by means of MGCPSO is left out in this subsection. The speed-up rates from all examples are discussed in a separate subsection.

5.3.1 Benchmark test

In this subsection, a benchmark test is implemented for problems (5.26) (5.6) selected from [4] are implemented by means of LPSO with K=2 and K=3 as well as MGCPSO. For convince of comparison, cross section area is used as design variable in the benchmark test, thus

problem P1 needs to be changed to

$$\begin{array}{lll} \text{P1} & \min\limits_{\mathbf{x}} & \mathbf{f}^T\mathbf{u} \\ & \text{subject to} & \sum\limits_{i=1}^{m} x_i l_i \mathsf{K}_i \mathbf{u} = \mathbf{f} \\ & & \sum\limits_{i=1}^{m} x_i l_i = V \\ & & x_i \geq 0, \quad i = 1, \cdots, m \end{array} \tag{5.28}$$

where x_i is the cross section area of member i. Note that in the case of integer problem (5.26) it is not possible to always satisfy the equality constraint of the total volume v. For this reason, it is relaxed to the following:

$$-1 \leq V - \sum_{i=1}^{m} x_i l_i \leq 1$$

These problems in this benchmark test stem from truss minimum compliance problems. Next, optimal solutions are presented and compared with those from [4] which prove to be the best results found so far. Each example is tested with four different kinds of design variables:

1. Member cross section area x_i is real and stays in the interval $[0,1]$, marked as $\mathbf{x} \in [0,1]^m$.
2. Member cross section area x_i is integer and has these candidates $\{0,1\}$, marked as $\mathbf{x} \in \{0,1\}^m$.
3. Member cross section area x_i is real and stays in the interval $[0,5]$, marked as $\mathbf{x} \in [0,5]^m$.
4. Member cross section area x_i is integer and has these candidates $\{0,\cdots,5\}$, marked as $\mathbf{x} \in \{0,5\}^m$.

where m is the number of design variables. The Young's modulus of elasticity E for all benchmark problems is scaled to unity for all bars as well as the external loads. All of the solutions obtained by means of LPSO are compared with those from [4] in figures.

5.3.1.1 A single-load wheel

The design domain, the load, as well as the boundary conditions are shown in figure 5.7(a) . A vertical load is applied at the center of the lower side of the design domain. The ground structure is shown in figure 5.7(b) . In addition, in order to achieve a stable solution, minute horizontal loads are applied to each design node. The optimal designs with continuous design variables $\mathbf{x} \in [0,1]^m$ obtained by LPSO with k=2 and k=3, as well as that from [4] are shown in figures 5.7(c) , 5.7(d) and 5.7(e) respectively. The solution from LPSO with k=2 is the best one, the solution from LPSO with k=3 is the second best one, however, the advantage is not obvious. The optimal designs with discrete design variables $\mathbf{x} \in \{0,1\}^m$ from

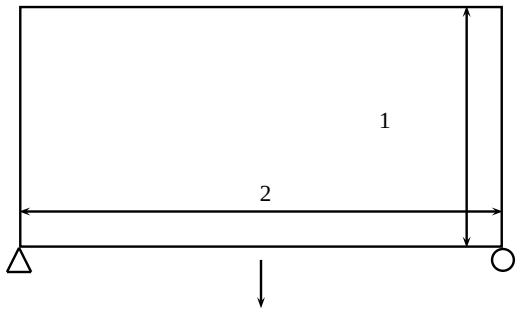
different optimization algorithms are the same and are shown in figure 5.7(f) . The optimal topologies of the continuous minimal compliance problem with $\mathbf{x} \in [0,5]^m$ from different algorithms are shown in figures 5.7(g) , 5.7(h) and 5.7(e) . Similarly, the LPSO with k=2 finds the best solution without obvious ascendancy. It must be noted that the solution in this instance from [4] is only stable in the vertical direction but a mechanism in other directions, so that, considering additional bars are used to guarantee structural stability which do not promote the objective function, the solutions from LPSO are more competitive. For the final example with discrete design variables $\mathbf{x} \in \{0,5\}^m$ solutions from LPSO with k=2 and k=3 are the same and are shown in figure 5.7(j) . Notably, it is a worse solution than that from [4] shown in figure 5.7(k) because of bars holding structural stability. The convergence curves for this problem are shown in figures 5.8(a) and 5.8(b) . The horizontal axis represents the generation of the penalty factor and the vertical axis shows the value of the corresponding penalized objective function, as below. It can be seen that the performance of the two differing LPSOs is quite similar with respect to real cases and virtually identical for integer problems. The convergence curves for integer cases are than those for the corresponding real cases due to a larger fitness value.

5.3.1.2 A single-load cantilever

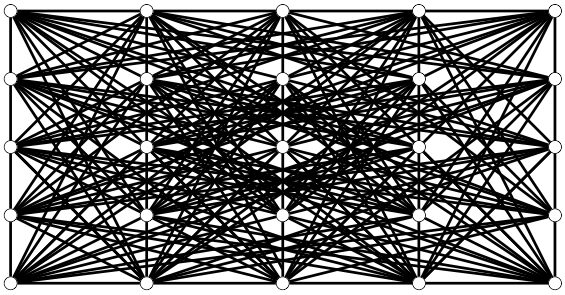
The design domain, external load and the boundary conditions for this cantilever example are shown in figure 5.9(a) . In this instance, a unit vertical load is applied at the lower right corner of the design domain. Its ground structure is shown in figure 5.9(b) . Similar to the first example, minute horizontal loads are applied to each design node in order to acquire a stable solution. The optimal designs with continuous design variables $\mathbf{x} \in [0,1]^m$ obtained by LPSO with k=2 and k=3, as well as that from [4] are shown in figures 5.9(c) , 5.9(d) and 5.9(e) respectively. The solutions from the LPSOs are stable both vertically and horizontally. Solution from LPSO with k=2 is slightly better than that from [4] where the bar suppressing the external load is a mechanism. Similar occurrences appear for problems with continuous design variables $\mathbf{x} \in [0,5]^m$ which is shown in figures 5.9(h) , 5.9(i) and 5.9(j) . For the problem with discrete design variables $\mathbf{x} \in \{0,1\}^m$ solutions from LPSO with k=2 and k=3 are the same and shown in figure 5.9(f) . It is notably worse than that from [4] shown in figure 5.9(l) because of the additional bars for upholding structural stability. In that sense it is similar to the solutions of the problem that has integer design variables $\mathbf{x} \in \{0,5\}^m$ by these two LPSOs. The convergence curves for this example are shown in figures 5.10(a) and 5.10(b) . It can be seen that the two different LPSOs exhibit a similar performance in the case of all four problems.

5.3.1.3 A single-load Michell beam example

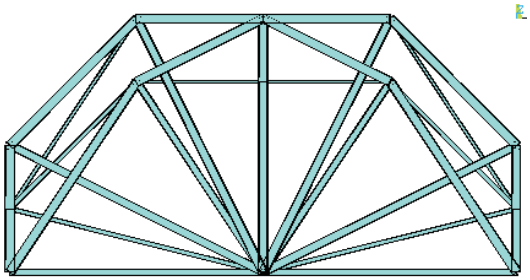
The design domain, the external load and the boundary conditions are illustrated in figure 5.11(a) . The vertical unit load is applied at the center of the right hand side of the design domain. Half of the left hand side is fixed to a wall. The ground structure for the design domain is shown in figure 5.11(b) . It is worth noting that this constitutes the largest problem



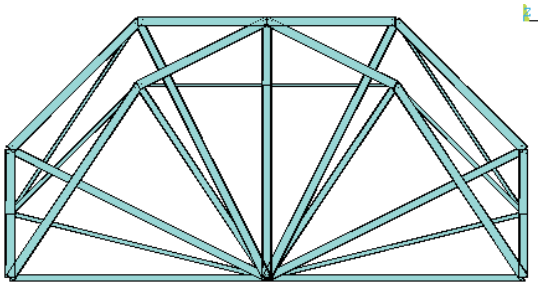
(a) The design domain



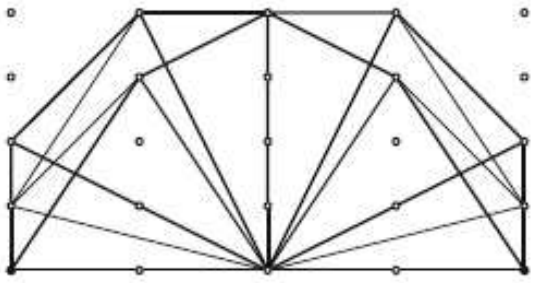
(b) The ground structure with 200 non-overlapping bars



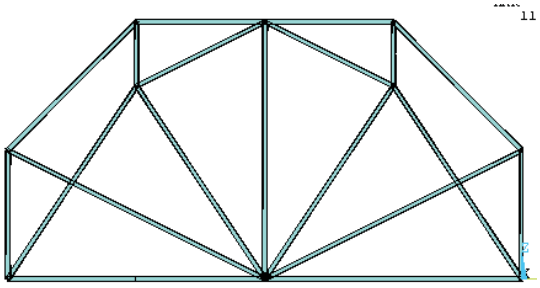
(c) A solution of the continuous optimization problem with $\mathbf{x} \in [0,1]^m$ and $V = 14$ solved by LPSO with $k = 2$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.4068066$



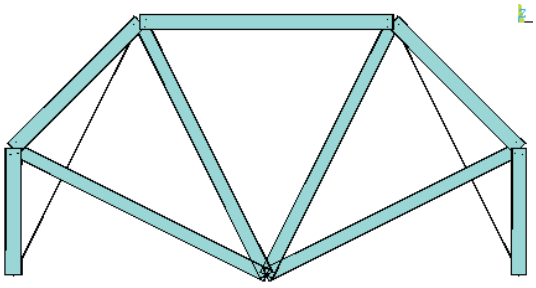
(d) A solution of the continuous optimization problem with $\mathbf{x} \in [0,1]^m$ and $V = 14$ solved by LPSO with $k = 3$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.4072573$



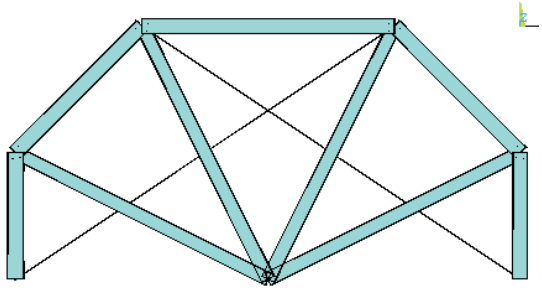
(e) A solution of the continuous optimization problem with $\mathbf{x} \in [0,1]^m$ and $V = 14$ from [4], $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.4070793$



(f) Solutions of the discrete problem with $\mathbf{x} \in \{0,1\}^m$ and $V = 14$ solved by LPSO with $K = 2$ or $K = 3$ are the same to that from [4] , $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.4462982$

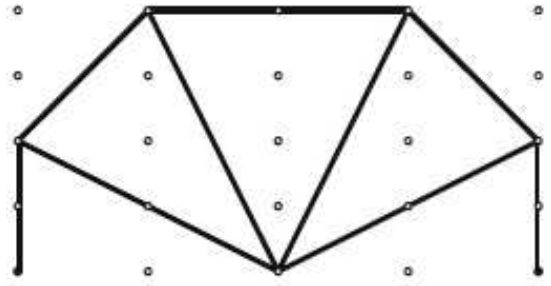


(g) A solution of the continuous optimization problem with $\mathbf{x} \in [0,5]^m$ and $V = 20$ solved by LPSO with $k = 2$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.2773669$

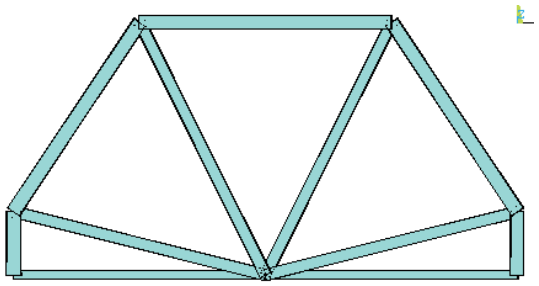


(h) A solution of the continuous optimization problem with $\mathbf{x} \in [0,5]^m$ and $V = 20$ solved by LPSO with $k = 3$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.2773722$

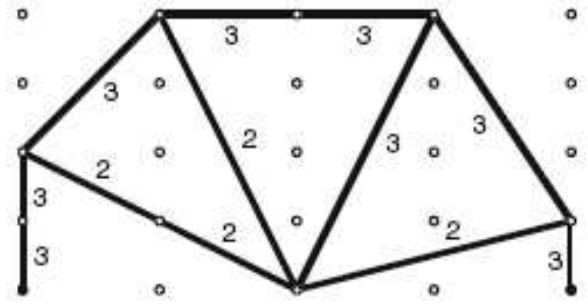
Figure 5.7: Summary of results from example 1



(i) A solution of the continuous optimization problem with $\mathbf{x} \in [0,5]^m$ and $V = 20$ from [4], $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.2777778$

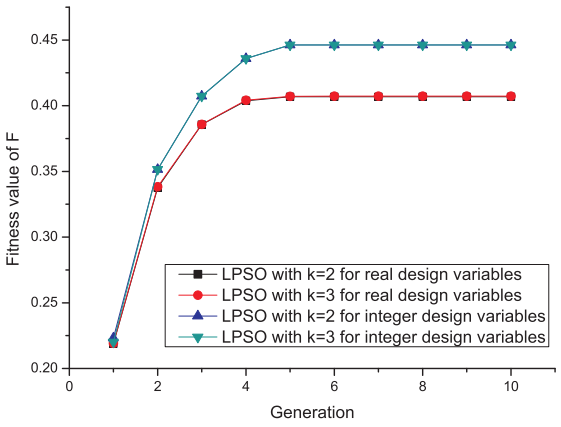


(j) Solutions of the discrete problem with $\mathbf{x} \in \{0,5\}^m$ and $V = 20$ solved by LPSO with $K = 2$ or $K = 3$ are the same, $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.3151256$

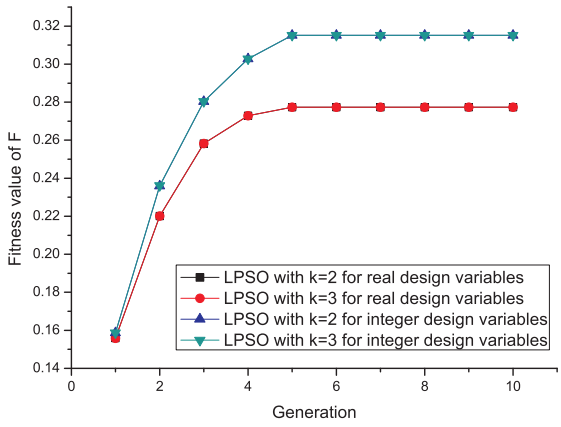


(k) A solution of the discrete problem with $\mathbf{x} \in \{0,5\}^m$ and $V = 20$ from [4], $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.2817373$

Figure 5.7: Summary of results from example 1 (cont.)

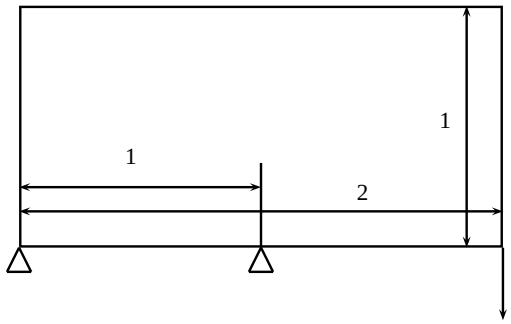


(a) converge curve for example 1 with $\mathbf{x} \in [0,1]^m$

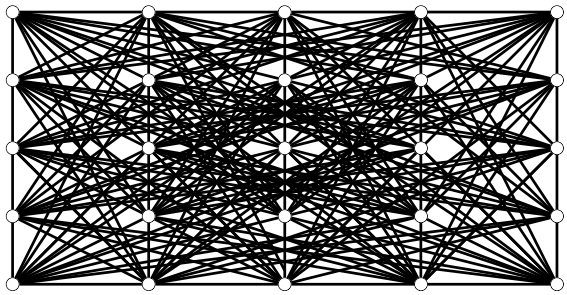


(b) converge curve for example 1 with $\mathbf{x} \in [0,5]^m$

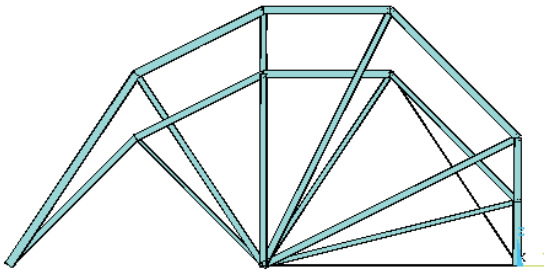
Figure 5.8: Converge curves for example 1



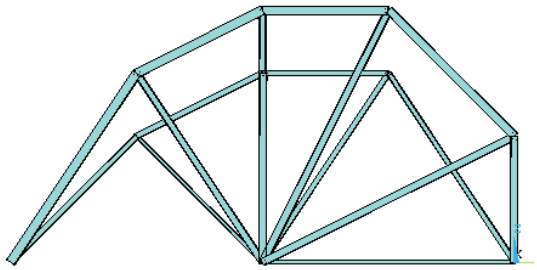
(a) The design domain



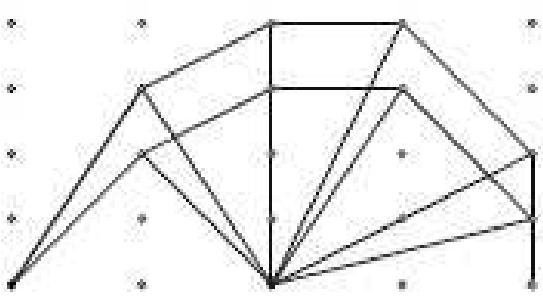
(b) The ground structure with 200 non-overlapping bars



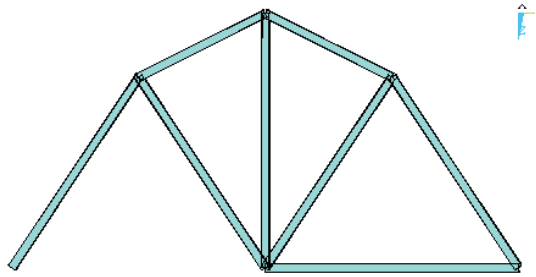
(c) A solution of the continuous optimization problem with $\mathbf{x} \in [0,1]^m$ and $V = 7$ solved by LPSO with $k = 2$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 2.642803$



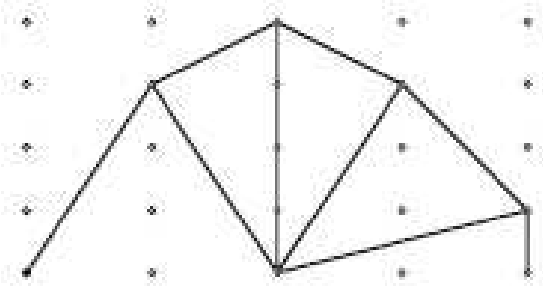
(d) A solution of the continuous optimization problem with $\mathbf{x} \in [0,1]^m$ and $V = 7$ solved by LPSO with $k = 3$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 2.654129$



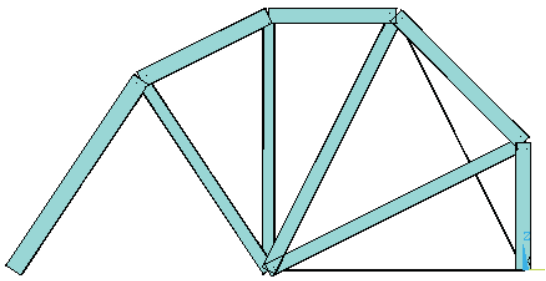
(e) A solution of the continuous optimization problem with $\mathbf{x} \in [0,1]^m$ and $V = 7$ from [4], $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 2.647288$



(f) Solutions of the discrete problem with $\mathbf{x} \in \{0,1\}^m$ and $V = 7$ solved by LPSO with $K = 2$ or $K = 3$ are the same, $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 3.066547$

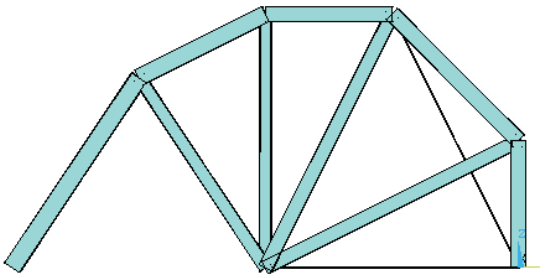


(g) A solution of the discrete problem with $\mathbf{x} \in \{0,1\}^m$ and $V = 7$ from [4], $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 2.999128$

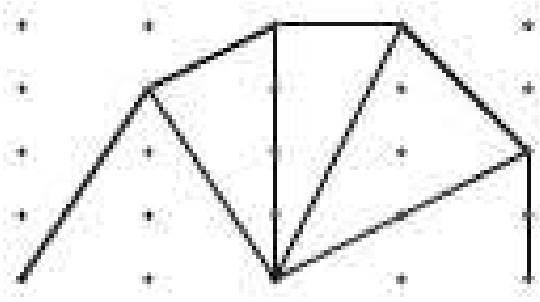


(h) A solution of the continuous optimization problem with $\mathbf{x} \in [0,5]^m$ and $V = 20$ solved by LPSO with $k = 2$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.9238297$

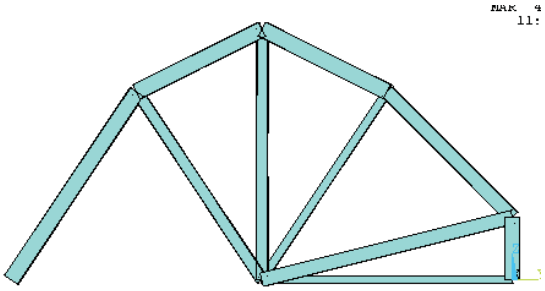
Figure 5.9: Summary of results from example 2



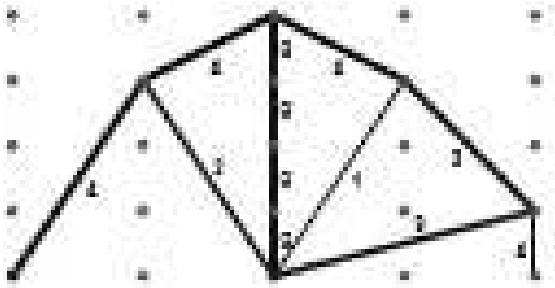
(i) A solution of the continuous optimization problem with $\mathbf{x} \in [0,5]^m$ and $V = 20$ solved by LPSO with $k = 2$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.9253831$



(j) A solution of the continuous optimization problem with $\mathbf{x} \in [0,5]^m$ and $V = 20$ from [4], $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.9251736$

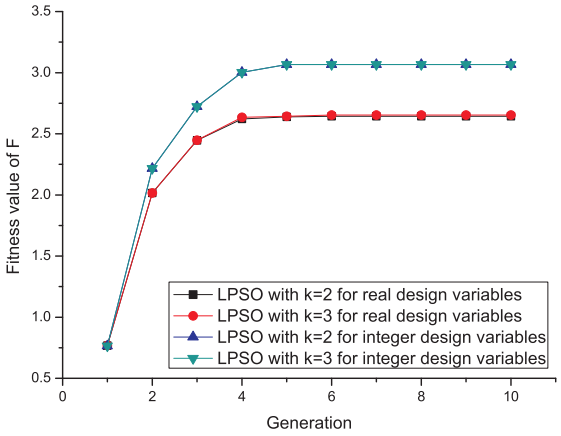


(k) Solutions of the discrete problem with $\mathbf{x} \in \{0,5\}^m$ and $V = 20$ solved by LPSO with $K = 2$ or $K = 3$ are the same, $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 1.026518$

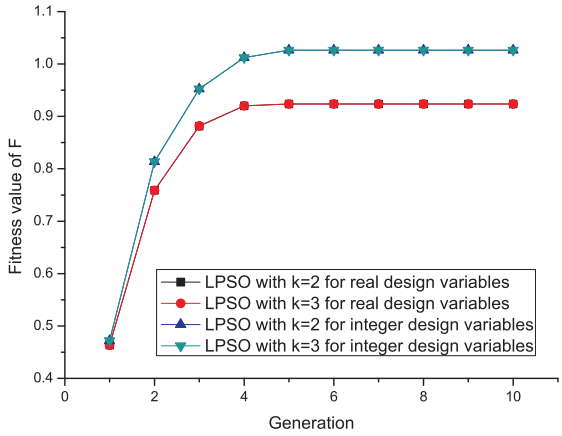


(l) A solution of the discrete problem with $\mathbf{x} \in \{0,5\}^m$ and $V = 20$ from [4], $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 0.9429963$

Figure 5.9: Summary of results from example 2 (cont.)



(a) Convergence curve for example 2 with $\mathbf{x} \in [0,1]^m$



(b) converge curve for example 2 with $\mathbf{x} \in [0,5]^m$

Figure 5.10: Convergence curves for example 2

dealt with in this benchmark test. Similar to the first and second examples, for the continuous optimization problems with real design variables $\mathbf{x} \in [0,1]^m$ and $\mathbf{x} \in [0,5]^m$, LPSO with k=2 obtained the best solution compared with the other two solutions. However, it is not in a dominating position. For the problem with integer design variables $\mathbf{x} \in \{0,1\}^m$ LPSO with k=2 again obtained the best solution. The solution from [4] is the second best one. If the design space is increased from $\mathbf{x} \in \{0,1\}^m$ to $\mathbf{x} \in \{0,5\}^m$, these two LPSOs found the same solution as [4]. The convergence curves for problems with $\mathbf{x} \in [0,1]^m$ and $\mathbf{x} \in [0,5]^m$ are shown in figure 5.12(a) and 5.12(b) respectively. It can be seen that the shapes of these curves are similar to those of the first and second example. This indicates that the quadratic penalty function works well with two LPSOs.

5.3.2 Further Examples

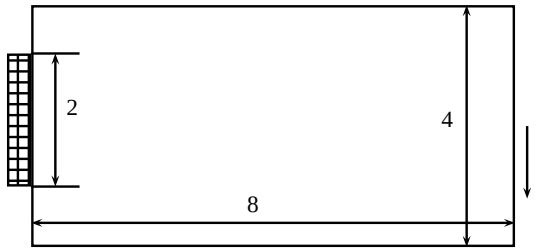
In subsection 5.3.1, the performance of two LPSOs is tested and competitive results are obtained. In order to expand application field of LPSO and MGCPSO, two supplementary examples are tested which are truss topology optimization with minimal weight and sizing optimization for steel structures. Compared with LPSP with k=2, LPSO with k=3 exhibits a fairly good performance within the lower number of iterations. LPSO with k=3 was only used in the first example.

5.3.2.1 Truss topology optimization

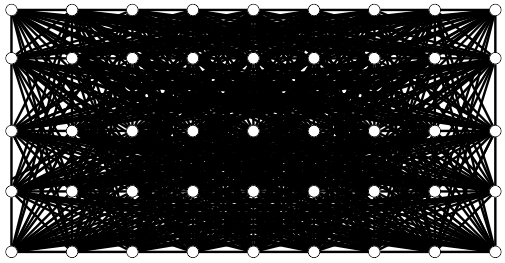
In this example, a truss is designed as a pedestrian bridge. The design domain is shown in figure 5.13(a) , distributed area load $p = 4\,\text{kPa}$ is applied on the upper surface, both ends of the design domain are restricted to joint-fixed bounds. Since this is a symmetric design problem, only half of the design is considered and the corresponding ground structure is shown in figure 5.13(b) . In order to avoid an unstable solution which is kinematic in the X direction, small external X-directional loads are applied to each design node. This is a 3-D example but only the possible connecting bars on the front and the upper surfaces are illustrated. Area loads are transformed into central loads which are applied to design nodes of the ground structure. The aim is to find a minimal volume structure that can withstand all structural constraints, including maximal deformation, allowed stresses, as well as local buckling. Only cross-section areas are used as design variables, while stresses and displacements are implicitly defined constraints that use the equilibrium equation. Local buckling is taken into account, meaning that if the ith bar is under compression then member stress must not exceed the Euler buckling stress which is given by

$$\sigma_i^{buck} = x_i \frac{\pi E_i}{4 l_i^2}$$

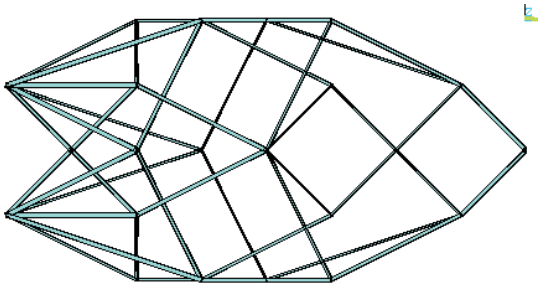
where $E_i = 206844\,\text{MPa}$ is the Young's modulus for bars and l_i is the ith bar's length. The maximal permissible deformation is set as $D_{max} = 25\,\text{mm}$. In order to make this example more practical, cross-section areas $\mathbf{x}$ are restricted to $[3225.8\,\text{mm}^2, 64516\,\text{mm}^2]$. Finally, this



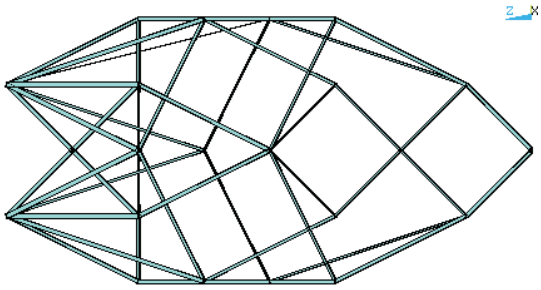
(a) The design domain



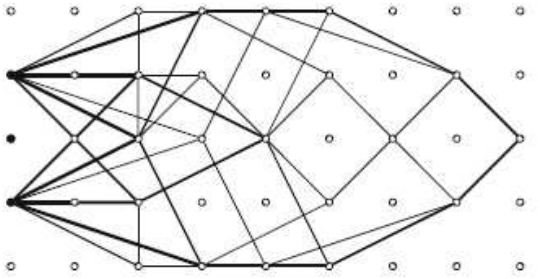
(b) The ground structure with 632 nonoverlapping bars



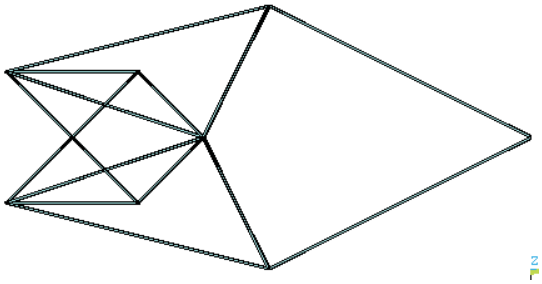
(c) A solution of the continuous optimization problem with $\mathbf{x} \in [0,1]^m$ and $V = 40$ solved by LPSO with $k = 2$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 21.97115$



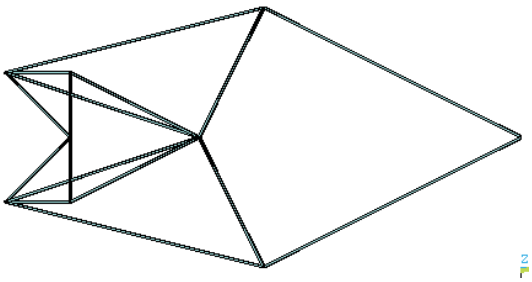
(d) A solution of the continuous optimization problem with $\mathbf{x} \in [0,1]^m$ and $V = 40$ solved by LPSO with $k = 3$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 21.97252$



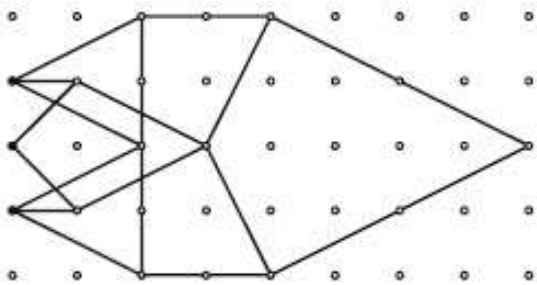
(e) A solution of the continuous optimization problem with $\mathbf{x} \in [0,1]^m$ and $V = 40$ from [4], $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 21.98687$



(f) A solution of the continuous optimization problem with $\mathbf{x} \in \{0,1\}^m$ and $V = 40$ solved by LPSO with $k = 2$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 24.54544$

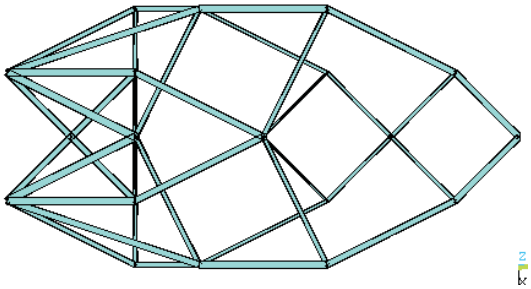


(g) A solution of the continuous optimization problem with $\mathbf{x} \in \{0,1\}^m$ and $V = 40$ solved by LPSO with $k = 3$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 25.30659$

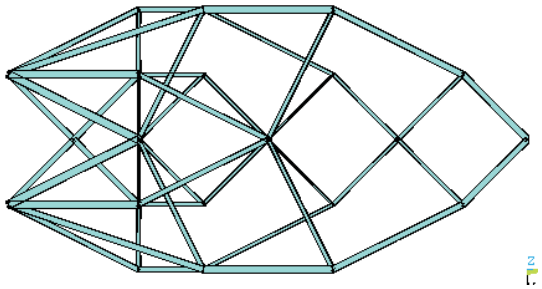


(h) A solution of the continuous optimization problem with $\mathbf{x} \in \{0,1\}^m$ and $V = 40$ from [4], $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 24.68202$

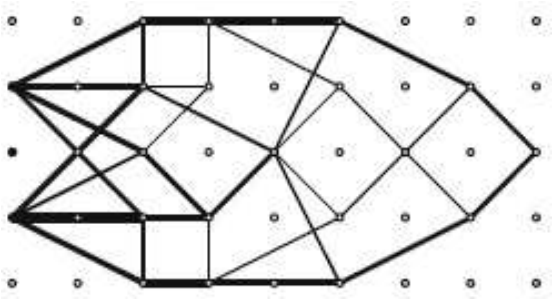
Figure 5.11: Summary of results from example 3



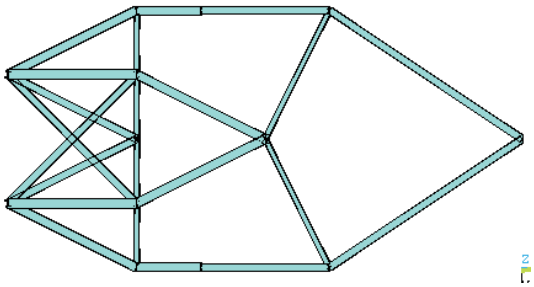
(i) A solution of the continuous optimization problem with $\mathbf{x} \in [0,5]^m$ and $V = 100$ solved by LPSO with $k = 2$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 8.771632$



(j) A solution of the continuous optimization problem with $\mathbf{x} \in [0,5]^m$ and $V = 100$ solved by LPSO with $k = 3$; $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 8.771661$

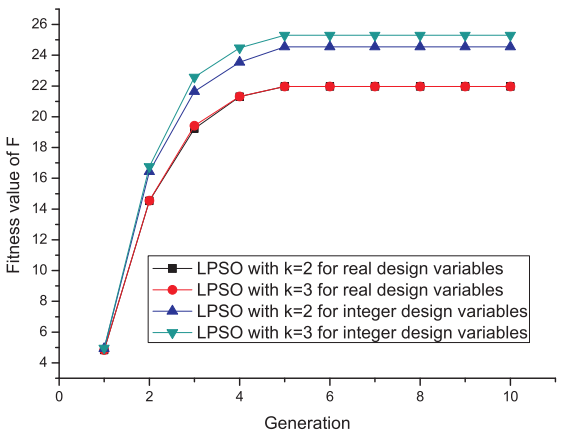


(k) A solution of the continuous optimization problem with $\mathbf{x} \in [0,5]^m$ and $V = 100$ from [4], $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 8.773395$

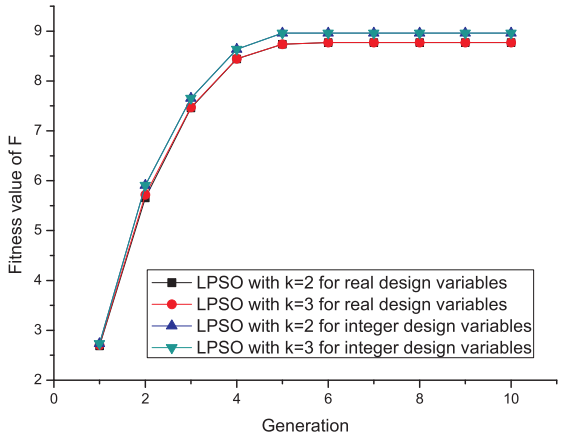


(l) Solutions of the discrete problem with $\mathbf{x} \in \{0,5\}^m$ and $V = 100$ solved by LPSO with $K = 2$ or $K = 3$ are the same to that from [4] , $\frac{1}{2}\mathbf{f}^T\mathbf{u} = 8.960962$

Figure 5.11: Summary of results from example 3 (cont.)



(a) Convergence curve for example 3 with $\mathbf{x} \in [0,1]^m$



(b) Convergence curve for example 3 with $\mathbf{x} \in [0,5]^m$

Figure 5.12: Convergence curves for example 3

optimization problem can be expressed as:

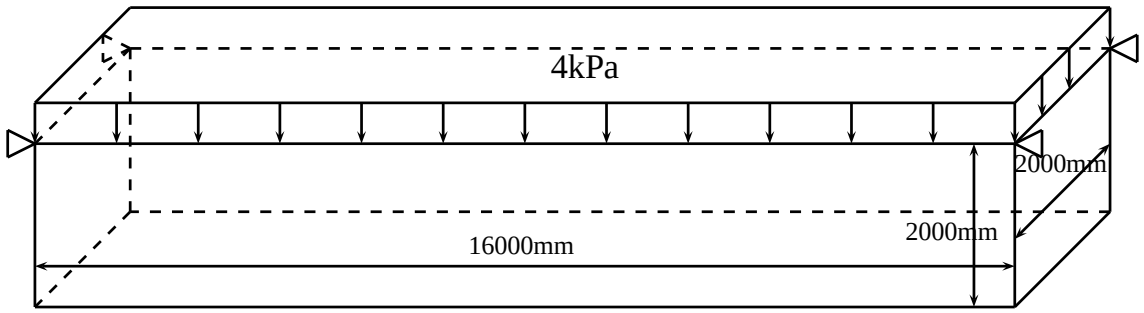
$$\begin{array}{ll} \min\limits_{\mathbf{x}} & \sum\limits_{i=1}^{n} x_i l_i \\ \text{subject to} & \mathsf{K}\mathbf{u} = \mathbf{f} \\ & D_{max} \leq 25 \\ & \mid \sigma_i \mid \leq 165.4752\,\text{MPa}, \quad i = 1, \cdots, n \\ & \sigma_i \leq \sigma_i^{buck}, \qquad\qquad \forall \sigma_i > 0, \quad i = 1, \cdots, n \end{array} \tag{5.29}$$

For this problem, formula (5.25) is used for handling constraints and the sequence of $\mu(k)$ is $10^k, k = \{4, 3, \cdots, -10\}$. The solutions obtained are shown in figures 5.13(c) , 5.13(d) , 5.13(e) and 5.13(f) , the volume of which is $7.8913D8\,\text{mm}^3$. It must be noted that this constitutes a reasonable structure. All of the vertical external loads are transformed to the boundary through diagonal bars on the front and the back surfaces. The diagonal bars inside the design body ensure that the structure is not a mechanism on the plane vertical to externals loads. Figures 5.13(c) and 5.13(g) show that all compressed bars are short and that their cross-section areas are larger than most bars in tension. This avoids local buckling and thus contributes to the stability of the structure. Although the connections on the upper surface are not continuous, maybe the absent bars can not promote structural stiffness, i.e. distributing them to another place contribute more to structural stiffness. The convergence curve is shown in figure 5.14 and it complies with the expected shape. The aforementioned demonstrates that the penalty approach of formula 5.25, together with PSO, is a useful tool in the dealing with structural constraints.

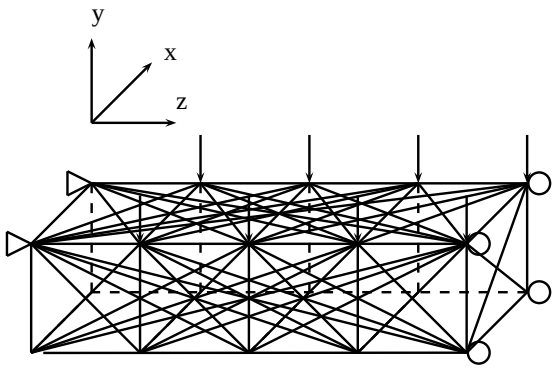
5.3.2.2 Structural sizing optimization

In order to extend the application field of MGCPSO, an elastic one-layer unbraced steel frame (as shown in figure 5.15(a)) is optimized by means of MGCPSO. The chinese design code for design of steel structures (GB 50017-2003) is applied. The cross section uses only I-section and the four corresponding design variables are shown in 5.15(b) . The construction material is Q235 steel, its strength is $f_y = 215\,\text{MPa}$, Young's modulus is $E = 206000\,\text{MPa}$ and the density is $\rho = 7800\,\text{Kg/m}^3$. As part of the standard procedure, prior to the implementation of structural optimization some simplicities need to be set up. In the case at hand, it is assumed that there are enough braces outside the design plane. Consequently, this frame is not in the danger of loosing stability outside the design plane. For this reason, there is no need for a stability check outside the design plane. The two beams use the same cross section. The column on the left hand side is the same as that on the right hand side. Since the horizontal load is small compared to the vertical one, it is assumed that shear stresses from columns are satisfied by the design code. Because the shear and axial forces for beams are small, no shear stress and stability constraint are applied to them. The total number of design unknowns is twelve and they belong to three different cross sections. In the interest of convenience, three important points are enumerated here:

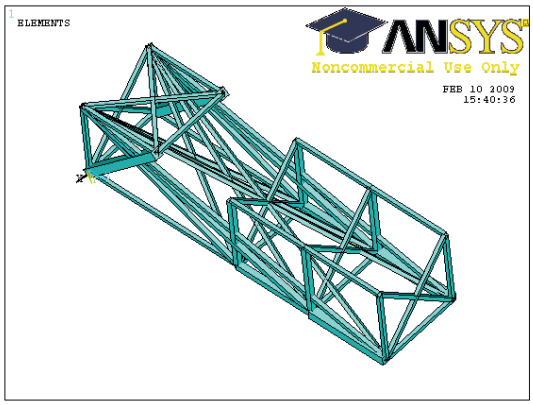
1. Effective length ratio μ: It is used to compute member's effective length and it should be selected from corresponding table supplied by the design code.



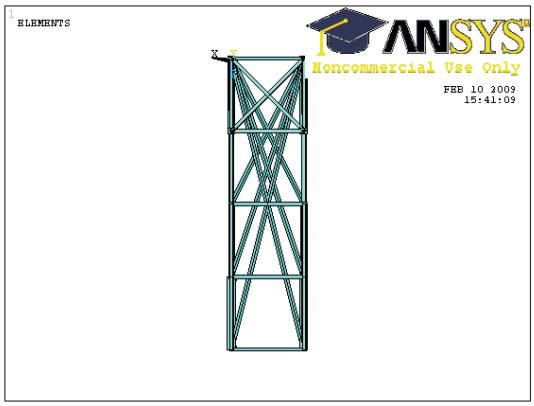
(a) Design domain



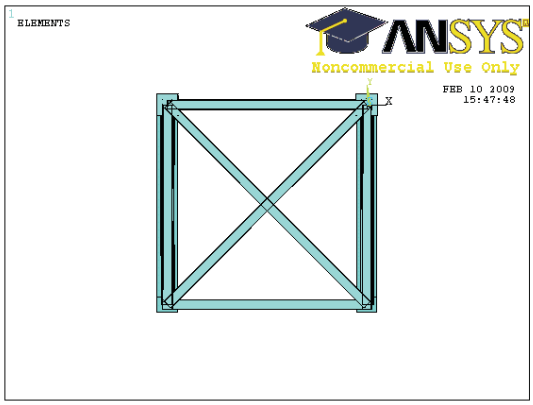
(b) Ground structure of half design domain



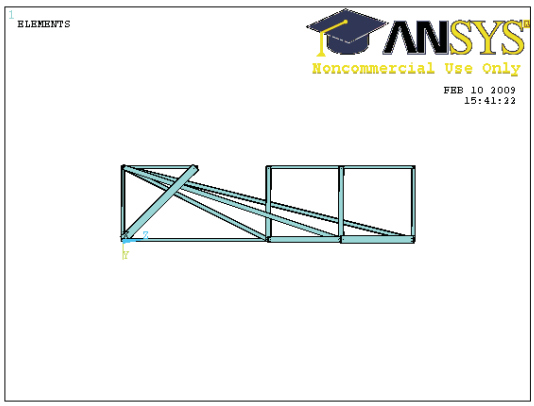
(c) Solution in isometric view



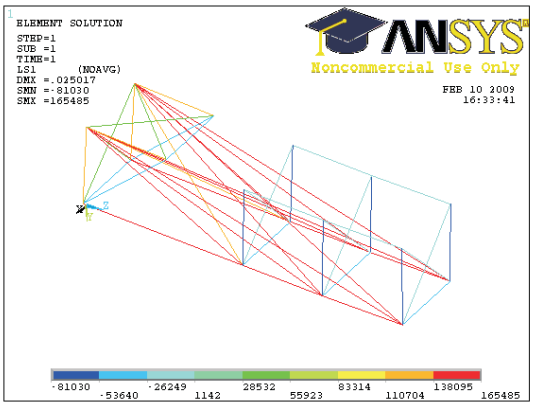
(d) Solution projected onto x-z plane



(e) Solution projected onto x-y plane



(f) Solution projected onto y-z plane



(g) Member stress

Figure 5.13: Summary of results from the first supplementary example

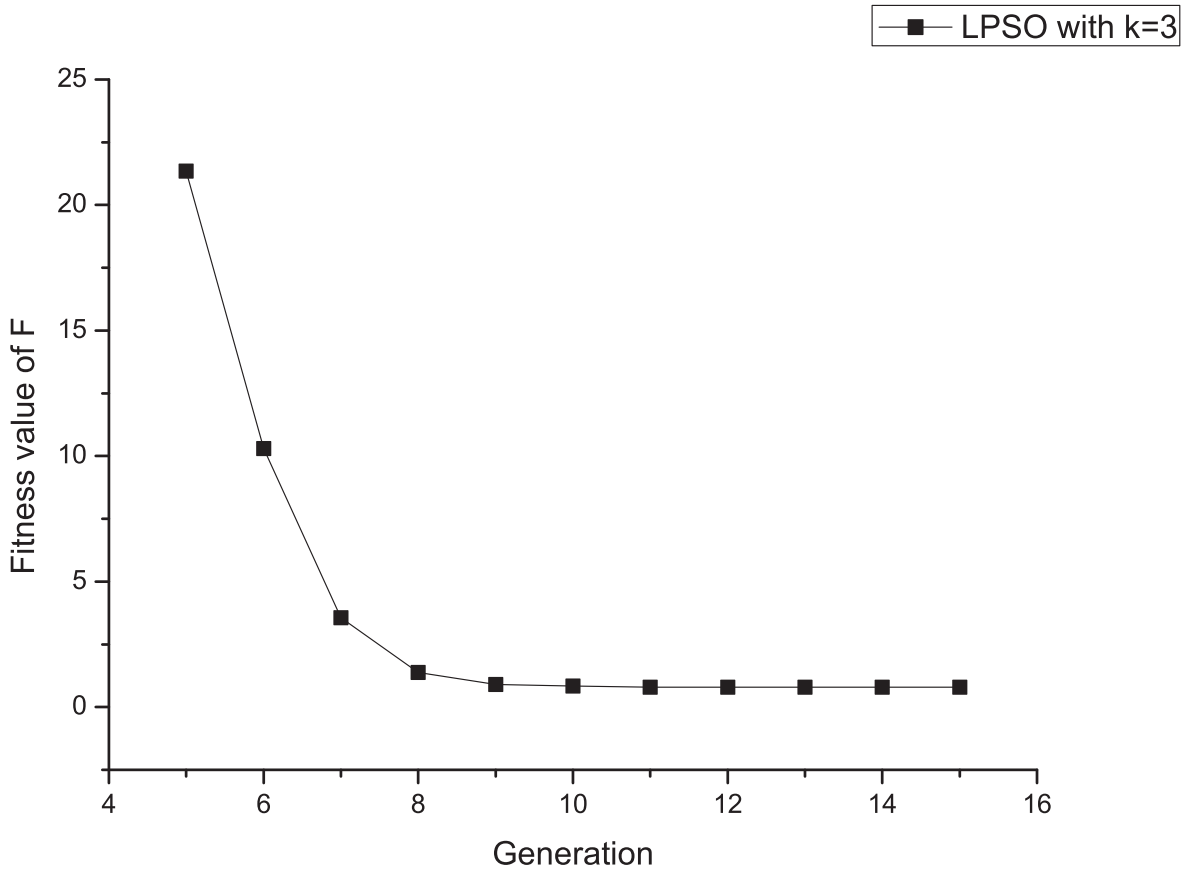


Figure 5.14: Convergence curve of supplementary example

2. Effective length l': It can be obtained by the $l' = \mu l$ where l is the member length.
3. Slenderness ratio λ: It can be obtained by $\lambda = l'/I_x$ where l' is effective length and I_x is the moment of inertia.

The optimization can now be stated as:

$$
\begin{array}{lll}
\min\limits_{\mathbf{h},\mathbf{b},\mathbf{t_1},\mathbf{t_2}} & f(\mathbf{h},\mathbf{b},\mathbf{t_1},\mathbf{t_2}) = \rho l_i \sum\limits_{i=1}^{3}(2h_i t_{2,i} + b_i t_{1,i}) & \\
\text{subject to} & \mathsf{K}\mathbf{u} = \mathbf{p} & \\
& u_{x,i} - 40\,\text{mm} \leq 0 & i = 4,5,6 \\
& \sigma_i - 215\,\text{MPa} \leq 0 & i = 1,2,3,4,5 \\
& \sigma_i^{stab} - 215\,\text{MPa} \leq 0 & i = 1,2,3,4,5 \\
& \lambda_i - 150 \leq 0 & i = 1,2,3 \\
& \dfrac{h_i - t_{1,i}}{t_{2,i}} - 15\sqrt{235/f_y} \leq 0 & i = 1,2,3 \\
& \dfrac{b}{t_{1,i}} - q \leq 0 & i = 1,2,3,4,5 \\
& 4\,\text{mm} \leq t_{1,i} \leq 16\,\text{mm} & i = 1,2,3 \\
& 4\,\text{mm} \leq t_{2,i} \leq 16\,\text{mm} & i = 1,2,3 \\
& 40\,\text{mm} \leq b_i \leq 800\,\text{mm} & i = 1,2,3 \\
& 40\,\text{mm} \leq h_i \leq 800\,\text{mm} & i = 1,2,3
\end{array}
\tag{5.30}
$$

The first equality constraint is the equilibrium equation. The second constraint is used to limit the maximal structural horizontal displacement. The third constraint is the normal strength constraint σ_i that is the maximum stress (normal stress plus bending stress) and it can be computed in the following terms:

$$\sigma_i = \frac{N_i}{A_i} + \frac{M_i}{w_{x,i}}$$

where A_i is the cross section area of member i, N_i is the axial direct force of member i, M_i is the maximal moment of member i and w_{xi} is the section modulus of member i. The fourth constraint is used to ensure structural in-plane stability and σ_i^{stab} and can be computed in terms of:

$$\sigma_i^{stab} = \frac{N_i}{\phi_{x,i} A_i} + \frac{M_i \beta_{mx,i}}{w_{x,i}(1 - 0.8 N_i / N_{EX,i})}$$

where $\beta_{mx,i}$ is the equivalent moment ratio and is set to 1.0 in this example. $N_{EX,i}$ can be computed as

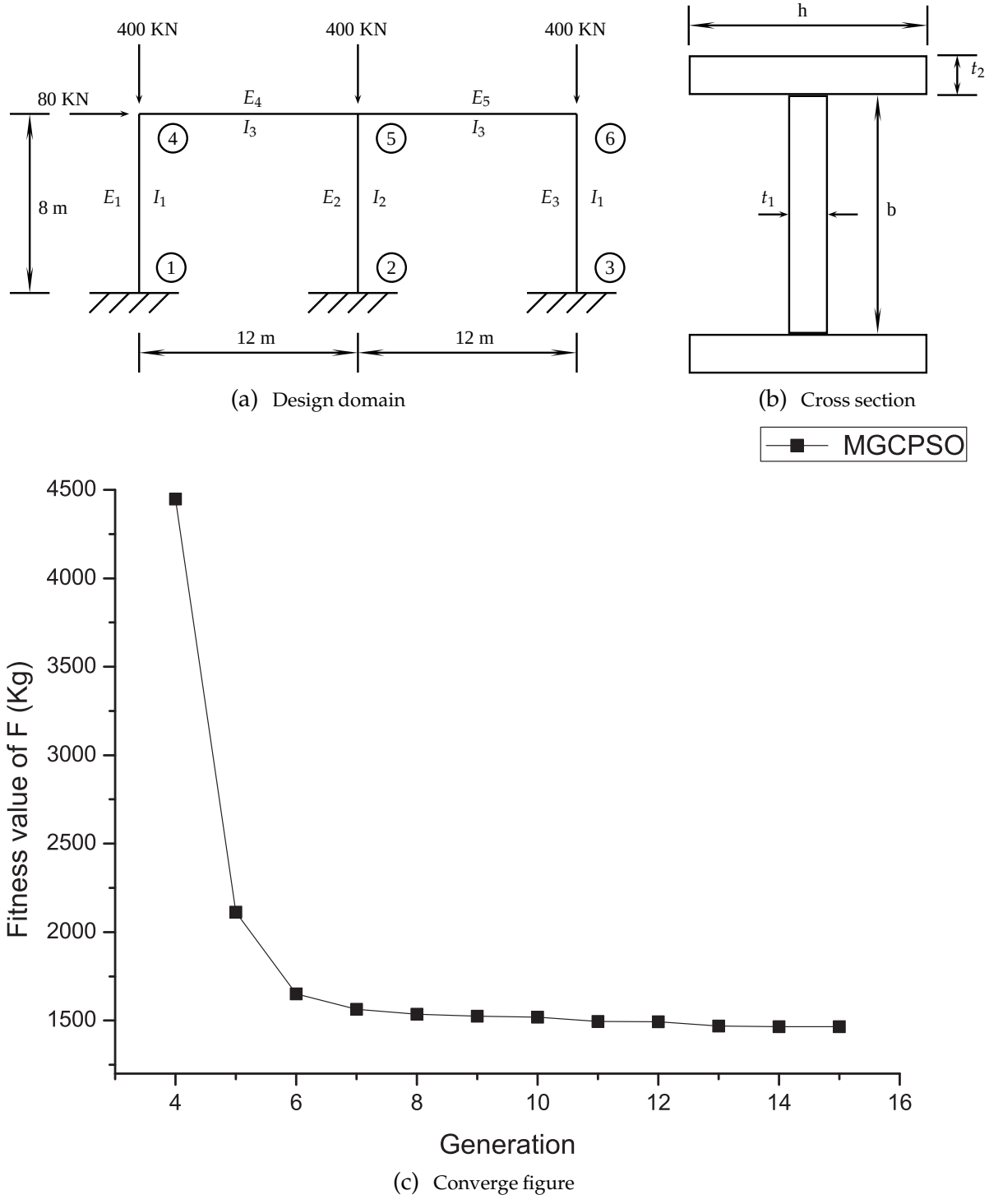
$$N_{EX,i} = \frac{\pi^2 E A_i}{1.1 \lambda_i^2}$$

The fifth constraint is for ensuring that the slenderness ratio is not too large. The sixth and the seventh constraints are used to protect members' flange and web local stabilities. q can be computed as

$$q = \begin{cases} (16a_0 + 0.5\lambda_i + 25)\sqrt{235/f_y} & 0 \leq a_0 \leq 1.6 \\ (48a_0 + 0.5\lambda_i - 26.2)\sqrt{235/f_y} & 1.6 < a_0 \leq 2.0 \end{cases}$$

with $a_0 = \frac{\sigma_{max} - \sigma_{min}}{\sigma_{max}}$. σ_{max} is the pressured stress at the end of flange, while σ_{min} is the stress on the other end. Note that pressured stress is assigned a positive value while tensile stress is negative. The last four constraints are the limits of design variables. In order to make this example more practical, integer programming is implemented. Rounding-off strategy discussed in subsection 2.5.5.2 is used here. It can be seen that this optimization contains various kinds of constraints, albeit the objective function being simple.

The optimized result for problem (5.30)is 1466.166 Kg. The optimum is listed in table 5.1. It can be seen that the center column is the largest member, because it is suppressed by the largest axial load and moment. The section areas of beams are smaller than that of columns since they aren't the main load-bearing members. Because this is a integer programming problem, the inequality constraints usually can not be exactly active. The nearly violated constraints are the third constraints on member M_1 and M_2 with $g_{3,1}(\mathbf{x}) = -3.266E-002$ and $g_{3,2}(\mathbf{x}) = -0.1512$, the fourth constraint on beam with $g_{4,3}(\mathbf{x}) = -0.5963$, as well as the sixth constraints on all columns with $g_{6,1}(\mathbf{x}) = -2.8352$, $g_{6,2}(\mathbf{x}) = -1.0890$ and $g_{6,3}(\mathbf{x}) = -2.2584$. It can now be concluded that for steel frame design, the normal strength constraints are usually satisfied and the critical strength constraints are combinational stresses concerning stability. The difference between these two kinds of stresses for the same member is more obvious with regard to columns. The local stability constraints



(a) Design domain (b) Cross section

(c) Converge figure

Figure 5.15: Structural sizing optimization

on flanges are also very important for columns. The convergence curve for this problem is shown in figure 5.15. This is a realistic curve which again proves that the quadratic penalty function works well with Particle Swarm Optimization algorithms, even for integer programming.

Section	t1	b	t2	h
I1	6	484	15	64
I2	10	746	16	84
I3	4	223	4	40

Table 5.1: Optimum of problem (5.30) (Unit: mm)

5.3.3 Computing effort and conclusion

The speed-up rates for the benchmark test and the first supplementary example are shown in figure 5.16. Because LPSO with k=2 and k=3 exhibit similar performance in parallel computing for the same kind of optimization problems, average speed-up rates are used here to represent the parallel efficiency. Note that the speed-up rates become worse as the scale of problems increases. The reason for this is that in case an intermediate structure is deemed infeasible by a geometry consistency check, the corresponding computing node will assign a value as large as its fitness value and wait until other computing nodes finish the structural analysis, as well as the evaluation of the fitness function. The waiting time coincides with a loss of parallel performance.

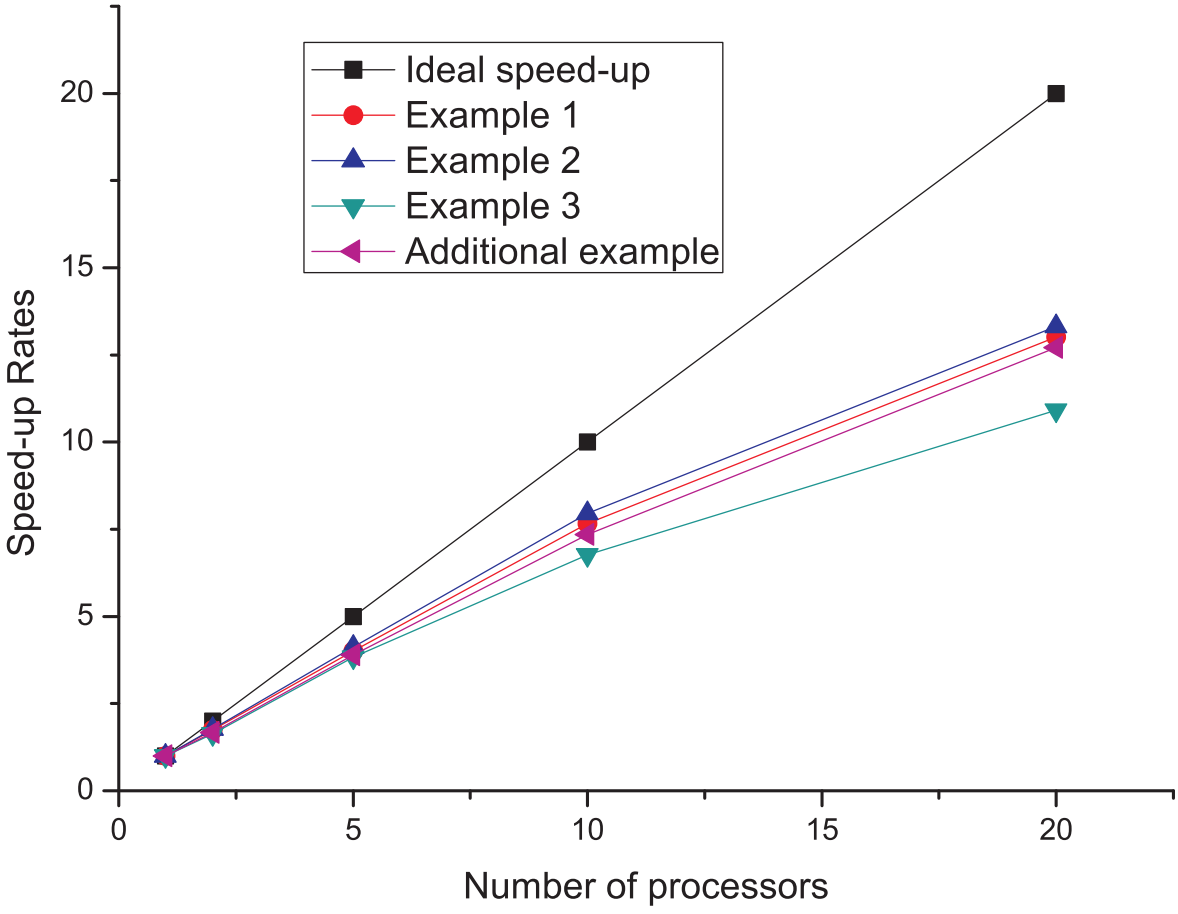


Figure 5.16: Average speed-up rates

Finally, it is concluded:

1. MGCPSO is not suited to large-scale, non-convex optimization problems (such as benchmark test), although it always converges faster than any LPSO. However, it was successfully applied to a structural sizing optimization problem.
2. LPSO with k=2 and k=3 exhibit fairly good global searching ability and obtain competitive results compared with those from [4]. This constitutes the best solution for the benchmark test so far.
3. LPSO can solve discrete truss topology optimization efficiently using the rounding-off strategy proposed in [76].
4. It proves that applying small external loads that are vertical to existing external loads is an effective way for obtaining a realistic structure.
5. The quadratic penalty function is proved effective, so long as it is combined with PSO.
6. To improve the speed-up rate, the parallel pattern needs to be modified or a heuristic geometry check implemented.

Chapter 6

Conclusion

This chapters outlines in brief the achievements of this thesis and is followed by a discussion of the directions that future work might take.

6.1 Summary

In chapter 3, two modified versions of PSO (MGCPSO and LPSO) are proposed, as well as their corresponding parallel patterns. MGCPSO is based on *Gbest* version of PSO, in turn, stems from PSO's *Lbest* version. The evaluation of these two modified varieties of PSO is based on the well-known and widely used benchmark test which contains characteristics that are difficult global optimization problem for evolutionary algorithms. A comparison is struck between the standard PSO solutions and those of the proposed innovations. On this basis, it can be concluded that both the MGCPSO, as well as the LPSO outperform the orthodox PSO and thus improve the performance of PSO in a meaningful way. It must be noted that it is the LPSO that achieved the best solution, albeit with a relatively low convergence speed. The MBCPSO, on the other hand, obtained a slightly worse solution but at a higher speed of convergence. In terms of the parallel performance of the parallel patterns, the MGCPSO achieves a better speed-up rate than the LPSO.

Chapter 4 contains a discussion of the application of MGCPSO to robust design, including also details concerning the self-updating model as proposed by Florian Jurecka [70]. The key advantage associated with the use of a numerical model instead of implementing complex computational analysis - such as finite element analysis - is the definite reduction of computing effort. The reason why reliance on the meta model updating procedure is essential is that the initial model is not able to accommodate the real characteristics of the original problem. In this work, the model updating strategy proposed by Florian Jurecka is implemented. The program is written in mixed Fortran-Matlab codes so as to allow the use of parallel computing without attracting the additional costs that are intrinsic to a strategy that relies on Matlab. The performance of MGCPSO is evaluated by means of the two numerical tests selected from [70] and DIRECT [48] is the chosen reference algorithm. It is interesting to note that MGCPSO and DIRECT not only arrive at nearly the same optimum but that it is relatively close to the real one also. Overall though, it is MGCPSO that exhibits the better performance out of the two approaches in this test. Largely due to the

strong performance of Fortran, MGCPSO achieves completion in a short amount of time. This advantage becomes even more obvious if a parallel version of MGCPSO is run.

In chapter 5, LPSO with k=2 and k=3 are applied to truss topology and structural sizing optimization. First, LPSO is applied to the benchmark test selected from [4]. Its objective is to determine a minimum compliance truss with a pre-determined volume, all of which can be stated as a problem (5.6). It is necessary to perform a geometry consistency check before the structural analysis in order to eliminate any redundant members and to choose possible intermediate structures since the LPSO relies on the problem in its original form 5.6 and searches the design space at random. In order to avoid obtaining a mechanism as a solution, it is essential to apply additional external loads to each design node of the ground structure. This method has shown itself to be both, very simple and effective. LPSO with k=2 and k=3 compares favourably even to the best solutions [4] obtained so far with regard to the benchmark test, as its optimal solutions for real, as well as integer optimization problems are competitive. The following example is concerned with the finding of a minimal weight truss structure which complies with all of the pre-determined structural constraints. Figures 5.13(c) -5.13(g) illustrate the reasonable solution found. Finally, MGCPSO is applied to a structural sizing optimization problem 5.30). What can be concluded from the convergence curves of all of the examples presented in this chapter is that the quadratic penalty function constitutes an effective method for handling structural constraints for PSO.

6.2 Further work

Despite having obtained successful results from all of the numerical tests, the room for further research is vast, including, amongst others, the following points of interest:

1. Make a converge proof for both variants so that they are able to maximise their potential by changing algorithm parameters or using adaptable parameters.
2. Examining the possibility of a new variant that results from forming a hybrid with the two proposed variants (MGCPSO and LPSO).
3. Applying MGCPSO to further problems of robust design by using a meta model and evaluate their performances.
4. Improve parallel efficiency by means of developing asynchronous parallel implementation pattern for LPSO or developing a heuristic method to make unanalyzable structure analyzable, since the loss of parallel performance is mainly caused by these unanalyzable intermediate structures.
5. Expanding LPSO to problems of truss topological optimization that feature more structural constraints (such as frequency, global stability and so on), problems of continuum material distribution, as well as those of material reinforcement. This is of interest because PSO still constitutes a considerably novel addition to the field topology optimization.

Bibliography

[1] MA Abido. Optimal power flow using particle swarm optimization. *International Journal of Electrical Power and Energy Systems*, 24(7):563–571, 2002.

[2] W. Achtziger, M. Bendsøe, A. Ben-Tal, and J. Zowe. Equivalent displacement based formulations for maximum strength Truss topology design. *IMPACT of Computing in Science and Engineering*, 4(4):315–345, 1992.

[3] W. Achtziger and M. Stolpe. Global optimization of truss topology with discrete bar areas Part II: Implementation and numerical results. *Computational Optimization and Applications*, pages 1–27.

[4] W. Achtziger and M. Stolpe. Truss topology optimization with discrete design variables-Guaranteed global optimality and benchmark examples. *Structural and Multidisciplinary Optimization*, 34(1):1–20, 2007.

[5] W. Achtziger and M. Stolpe. Global optimization of truss topology with discrete bar areas Part I: theory of relaxed problems. *Computational Optimization and Applications*, 40(2):247–280, 2008.

[6] D. H. Ackley. *A connectionist machine for genetic hillclimbing*. Kluwer, Boston, 1987.

[7] J.E. Alvarez-Benitez, R.M. Everson, and J.E. Fieldsend. A MOPSO Algorithm Based Exclusively on Pareto Dominance Concepts. In *Evolutionary Multi-criterion Optimization:... International Conference, EMO...: Proceedings*. Springer, 2001.

[8] P.J. Angeline. Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 601–610, 1998.

[9] PJ Angeline, N.S. Inc, and NY Vestal. Using selection to improve particle swarm optimization. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 84–89, 1998.

[10] P. Audze and V. Eglais. New approach to planning out of experiments. *Problems of dynamics and strength*, 35:104–107, 1977.

[11] T. Bäck. *Evolutionary algorithms in theory and practice*. Oxford University Press, 1996.

[12] T. Back and H.P. Schwefel. An overview of evolutionary algorithmsfor parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[13] PD Beattie and JM Bishop. Self-Localisation in the Senario Autonomous Wheelchair. *Journal of Intelligent and Robotic Systems*, 22(3):255–267, 1998.

[14] M. Beekers and C. Fleury. A primal-dual approach in truss topology optimization. *Computers and Structures*, 64(1-4):77–88, 1997.

[15] A. Ben-Tal and M.P. Bendsøe. A New Method for Optimal Truss Topology Design. *SIAM Journal on Optimization*, 3:322, 1993.

[16] A. Ben-Tal and A. Nemirovskii. Potential Reduction Polynomial Time Method for Truss Topology Design. *SIAM Journal on Optimization*, 4:596, 1994.

[17] A. Ben-Tal and M. Zibulevsky. Penalty/Barrier Multiplier Methods for Convex Programming Problems. *SIAM JOURNAL OF OPTIMIZATION*, 7:347–366, 1997.

[18] MP Bendsøe, A. Ben-Tal, and J. Zowe. Optimization methods for truss geometry and topology design. *Structural and Multidisciplinary Optimization*, 7(3):141–159, 1994.

[19] S. Bird and X. Li. Adaptively choosing niching parameters in a PSO. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 3–10. ACM New York, NY, USA, 2006.

[20] JM Bishop. Stochastic Searching Networks. In *Proc. 1 stIEE Conf. Artificial Neural Networks*, pages 329–331.

[21] JM Bishop and P. Torr. THE STOCHASTIC SEARCH NETWORK. *Neural Networks for Vision, Speech, and Natural Language*, 1992.

[22] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, USA, 1999.

[23] J. Branke and S. Mostaghim. About Selecting the Personal Best in Multi-Objective Particle Swarm Optimization. *LECTURE NOTES IN COMPUTER SCIENCE*, 4193:523, 2006.

[24] R. Brits, A.P. Engelbrecht, and F. van den Bergh. A Niching Particle Swarm Optimizer. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL02)*, volume 2, pages 692–696.

[25] D.F. Carvalho and C.J.A. Bastos-Filho. Clan Particle Swarm Optimization. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3044–3051, 2008.

[26] HN Chen, YL Zhu, KY Hu, and T. Ku. Global optimization based on hierarchical coevolution model. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1497–1504, 2008.

[27] S. Christensen and F. Oppacher. What can we learn from no free lunch? a first attempt to characterize the concept of a searchable function. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1219–1226, 2001.

[28] M. Clerc. The swarm and the queen: towards a deterministic and adaptiveparticle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, 1999.

[29] M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in amultidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.

[30] 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*, 191(11-12):1245–1287, 2002.

[31] Z. Cui and J. Zeng. A Guaranteed Global Convergence Particle Swarm Optimizer. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 762–767, 2004.

[32] G.B. Dantzig. *Linear Programming and Extensions*. Princeton Univ Pr, 1963.

[33] KA De Jong. An analysis of the behavior of a class of genetic adaptive systems (Doctoral dissertation, university of Michigan). *Dissertation Abstracts International*, 36(10):76–9381, 1975.

[34] K. De Meyer, JM Bishop, and SJ Nasuto. STOCHASTIC DIFFUSION: USING RECRUITMENT FOR SEARCH. *Evolvability and interaction: evolutionary substrates of communication, signalling, and perception in the dynamics of social complexity (ed. P. McOwan, K. Dautenhahn & CL Nehaniv) Technical Report*, 393:60–65, 2003.

[35] Van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, Pretoria, 2002.

[36] M. Dorigo. Optimization, Learning and Natural Algorithms. *PhDthesis, PolitecnicodiMilano*, 1992.

[37] W.S. Dorn, R.E. Gomory, and H.J. Greenberg. Automatic design of optimal structures. *Journal de Mecanique*, 3(6):2552, 1964.

[38] R.C. Eberhart and J.F. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the sixth International Symposium on Micro Machine and Human Science*, pages 39–43, Nagoya, Japan, 1995.

[39] Simpson P. Eberhart, R.C. and R. Dobbins. *Computational Intelligence PC Tools*. Academic Press, 1996.

[40] I. EILISHAKOFF, RT Haftka, and J. Fang. Structural design under bounded uncertainty-optimization with anti-optimization. *Computers & structures*, 53(6):1401–1405, 1994.

[41] AA EL-Dib, HKM Youssef, MM EL-Metwally, and Z. Osman. Load flow solution using hybrid particle swarm optimization. In *Electrical, Electronic and Computer Engineering, 2004. ICEEC'04. 2004 International Conference on*, pages 742–746, 2004.

[42] AP Engelbrecht and LNH van Loggerenberg. Enhancing the NichePSO. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 2297–2302, 2007.

[43] L. Eriksson, E. Johansson, N. Kettaneh-Wold, C. Wikström, and S. Wold. *Design of Experiments: Principles and Applications*. Umetrics Academy, 2000.

[44] HY FAN and Y. SHI. Study of Vmax of the particle swarm optimization algorithm. In *the Workshop on Particle Swarm Optimization, Indianapolis, IN: Purdue School of Engineering and Technology*, 2001.

[45] S.S. Fan, Y. Liang, and E. Zahara. Hybrid simplex search and particle swarm optimization for the global optimization of multimodal functions. *Engineering Optimization*, 36(4):401–418, 2004.

[46] A.V. Fiacco and G.P. McCormick. The Sequential Unconstrained Minimization Technique for Nonlinear Programing, a Primal-Dual Method. *Management Science*, pages 360–366, 1964.

[47] A.V. Fiacco and G.P. McCormick. *Nonlinear programming: sequential unconstrained minimization techniques*. Society for Industrial Mathematics, 1990.

[48] D.E. Finkel. Direct optimization algorithm user guide. *Center for Research in Scientific Computation North Carolina State University, Raleigh, NC*, pages 27695–8205, 2003.

[49] P. Fleron. The minimum weight of trusses. *Bygningsstatiske Meddelelser*, 35(3):81–96, 1964.

[50] A. Flexer. Statistical Evaluation of Neural Network Experiments: Minimum Requirements and Current Practice. *CYBERNETICS AND SYSTEMS RESEARCH*, pages 1005–1008, 1996.

[51] L.J. Fogel. *On the Organization of Intellect.* PhD thesis, University of California, Los Angeles-Engineering, 1964.

[52] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons Inc, 1966.

[53] R.M. Friedberg. A learning machine: Part I. *IBM Journal of Research and Development*, 2(1):2–13, 1958.

[54] RM Friedberg, B. Dunham, and JH North. A learning machine. II. *IBM Journal of Research and Development*, 3:282–287, 1959.

[55] B. G.M.Husslage. *Maximin designs for computer experiments*. PhD thesis, Tilburg University,NL, 2006.

[56] E.M. GORDON. Cramming more components onto integrated circuits. *Intel Corporation ftp://download. intel. com/research/silicon/moorespaper. pdf*, 1965.

[57] HJ Grech-Cini and G.T. McKee. Locating the mouth region in images of human faces. In *Proceedings of SPIE*, volume 2059, page 458. SPIE, 1993.

[58] P. HAJELA and E. LEE. Genetic algorithms in truss topological optimization. *International journal of solids and structures*, 32(22):3341–3357, 1995.

[59] Q. He and C. Han. An Improved Particle Swarm Optimization Algorithm with Disturbance Term. *Jisuanji Gongcheng yu Yingyong(Computer Engineering and Applications)*, 43(7):84–86, 2007.

[60] S. He, QH Wu, JY Wen, JR Saunders, and RC Paton. A particle swarm optimizer with passive congregation. *BioSystems*, 78(1-3):135–147, 2004.

[61] T. Hendtlass. Preserving diversity in particle swarm optimisation. In *Proceedings of the 16th international conference on Developments in applied artificial intelligence*, pages 31–40. Springer Springer Verlag Inc, 2003.

[62] F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. *AMERICAN ASSOCIATION FOR THE ADVANCEMENT OF SCIENCE, WASHINGTON, DC(USA). 1990.*, 1990.

[63] N. Higashi and H. Iba. Particle swarm optimization with Gaussian mutation. In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pages 72–79, 2003.

[64] JH Holland. Adaptation in Natural and Artificial System: An Introduction with Application to Biology, Control and Artificial Intelligence. *Ann Arbor, University of Michigan Press*, 1975.

[65] X. Hu and RC Eberhart. Adaptive particle swarm optimization: detection and response todynamic systems. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, 2002.

[66] S. Hurley and R.M. Whitaker. An agent based approach to site selection for wireless networks. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 574–577. ACM New York, NY, USA, 2002.

[67] R.L. Iman, JC Helton, and J.E. Campbell. An approach to sensitivity analysis of computer models, Part I. Introduction, input variable selection and preliminary variable assessment. *Journal of Quality Technology*, 13(3):174–183, 1981.

[68] F. Jarre, M. Kocvara, and J. Zowe. Optimal Truss Design by Interior-Point Methods. *SIAM JOURNAL OF OPTIMIZATION*, 8:1084–1107, 1998.

[69] D.R. Jones, M. Schonlau, and W.J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[70] F. Jurecka. *Robust Design Optimization Based on Metamodeling Techniques*. PhD thesis, Technical University Munich, Munich, Germany, 2007.

[71] A. Kaveh and S. Talatahari. A HYBRID PARTICLE SWARM AND ANT COLONY OPTIMIZATION FOR DESIGN OF TRUSS STRUCTURES. *ASIAN JOURNAL OF CIVIL ENGINEERING (BUILDING AND HOUSING)*, 9(4):329–348, 2008.

[72] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, 1995.

[73] J. Kennedy and RC Eberhart. A discrete binary version of the particle swarm algorithm. In *Systems, Man, and Cybernetics, 1997.'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on*, volume 5, 1997.

[74] J.F. Kennedy, R.C. Eberhart, Y. Shi, and ScienceDirect (Online service. *Swarm intelligence*. Springer, 2001.

[75] D. G. Krige. A statistical approach to some mine valuation and allied problems on the witwatersrand. Master's thesis, University of the Witwatersrand, South Africa, 1951.

[76] EC Laskari, KE Parsopoulos, and MN Vrahatis. Particle swarm optimization for integer programming. In *Proceedings of the IEEE 2002 Congress on Evolutionary Computation*, pages 1576–1581, 2002.

[77] T. Lewiński and G.I.N. Rozvany. Exact analytical solutions for some popular benchmark problems in topology optimization II: three-sided polygonal supports. *Structural and Multidisciplinary Optimization*, 33(4):337–349, 2007.

[78] T. Lewiński and G.I.N. Rozvany. Exact analytical solutions for some popular benchmark problems in topology optimization III: L-shaped domains. *Structural and Multidisciplinary Optimization*, 35(2):165–174, 2008.

[79] T. Lewiński, M. Zhou, and GIN Rozvany. Exact least-weight truss layouts for rectangular domains with various support conditions. *Structural and Multidisciplinary Optimization*, 6(1):65–67, 1993.

[80] X. Li. Adaptively Choosing Neighbourhood Bests Using Species in a Particle Swarm Optimizer for Multimodal Function Optimization. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 105–116, 2004.

[81] X. Li, J. Branke, and T. Blackwell. Particle swarm with speciation and adaptation in a dynamic environment. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 51–58. ACM New York, NY, USA, 2006.

[82] Y. Liu and KM Passino. Biomimicry of Social Foraging Bacteria for Distributed Optimization: Models, Principles, and Emergent Behaviors. *Journal of Optimization Theory and Applications*, 115(3):603–628, 2002.

[83] A. Loengarov and V. Tereshko. A minimal model of honey bee foraging. In *Proc. IEEE Swarm Intell. Symp*, pages 175–182, 2006.

[84] M. Lombardi and R.T. Haftka. Anti-optimization technique for structural design under load uncertainties. *Computer methods in applied mechanics and engineering*, 157(1-2):19–31, 1998.

[85] M. Løvberg. Improving Particle Swarm Optimization by Hybridization of Stochastic Search Heuristics and Self Organized Critically, Master's Thesis. *Department of Computer Science, University of Aarhus, Denmark*, 2002.

[86] M. Løvbjerg. Improving particle swarm optimization by hybridization of stochastic search heuristics and self-organized criticality. 2002.

[87] G. Matheron. Principles of geostatistics. *Economic Geology*, 58(8):1246–1266, 1963.

[88] MD McKay, RJ Beckman, and WJ Conover. A comparison of three methods for selecting values of input variables in the analysis of out put from a computer code [J]. *Technometrics*, 21(2):239–245, 1979.

[89] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: simpler, maybe better. *Evolutionary Computation, IEEE Transactions on*, 8(3):204–210, 2004.

[90] AGM Michell. The limits of economy of material in frame structures. *Phil. Mag*, 8(47):589–597, 1904.

[91] M.M. Millonas. A nonequilibrium statistical field theory of swarms and other spatially extended complex systems. *eprint arXiv: adap-org/9306001*, 1993.

[92] V. Miranda and N. Fonseca. EPSO-evolutionary particle swarm optimization, a new algorithm with applications in power systems. In *Proc. of the Asia Pacific IEEE/PES Transmission and Distribution Conference and Exhibition*, volume 2, pages 745–750, 2002.

[93] A.S. Mohais, R. Mendes, C. Ward, and C. Posthoff. Neighborhood Re-structuring in Particle Swarm Optimization. *LECTURE NOTES IN COMPUTER SCIENCE*, 3809:776, 2005.

[94] AS Mohais, C. Ward, and C. Posthoff. Randomized directed neighborhoods with edge migration in particle swarm optimization. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, 2004.

[95] S. Mostaghim and J. Teich. Strategies for finding good local guides in multi-objective particle swarm optimization (MOPSO). In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pages 26–33, 2003.

[96] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, 1999.

[97] M. Ohsaki and C.C. Swan. Topology and Geometry Optimization of Trusses and Frames. *Recent Advances in Optimal Structural Design*, 2002.

[98] A.B. Owen. Orthogonal arrays for computer experiments, integration and visualization. *Statistica Sinica*, 2(2):439–452, 1992.

[99] E. Ozcan and CK Mohan. Particle swarm optimization: surfing the waves. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, 1999.

[100] N. Padhye. Topology optimization of compliant mechanism using multi-objective particle swarm optimization. In *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pages 1831–1834. ACM New York, NY, USA, 2008.

[101] RE Perez and K. Behdinan. Particle swarm approach for structural design optimization. *Computers and Structures*, 85(19-20):1579–1588, 2007.

[102] R. Poli. An analysis of publications on particle swarm optimization applications. Technical report, Department of Computing and Electronic Systems, University of Essex, Colchester, Essex, UK, 2007.

[103] W. Prager. *Introduction to structural optimization*. Springer.

[104] A. Ratnaweera, SK Halgamuge, and HC Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *Evolutionary Computation, IEEE Transactions on*, 8(3):240–255, 2004.

[105] I. Rechenberg. Cybernetic solution path of an experimental problem. *Library Translation*, 1122, 1964.

[106] I. Rechenberg. Evolution strategy: optimization of technical systems according to the principles of biological evolution. *Frommann-Holzboog, Stuttgart*, 1973.

[107] C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34. ACM Press New York, NY, USA, 1987.

[108] C.W. Reynolds. Flocks, herds, and schools: a distributed behavioural model. *Computer Garphics*, 21(4):25–34, 1987.

[109] J. Riget and J.S. Vesterstrøm. A diversity-guided particle swarm optimizer-the ARPSO. *Dept. Comput. Sci., Univ. of Aarhus, Aarhus, Denmark, Tech. Rep*, 2:2002, 2002.

[110] U.T. Ringertz. A BRANCH AND BOUND ALGORITHM FOR TOPOLOGY OPTIMIZATION OF TRUSS STRUCTURES. *Engineering Optimization*, 10(2):111–124, 1986.

[111] GIN Rozvany. Structural design via optimality criteria. *Mechanics of elastic a. inelastic solids; N8*, 1989.

[112] GIN Rozvany. Difficulties in truss topology optimization with stress, local buckling and system stability constraints. *Structural and Multidisciplinary Optimization*, 11(3):213–217, 1996.

[113] GIN Rozvany. Exact analytical solutions for some popular benchmark problems in topology optimization. *Structural and Multidisciplinary Optimization*, 15(1):42–48, 1998.

[114] W.K. Rule. Automatic Truss Design by Optimized Growth. *Journal of Structural Engineering*, 120(10):3063–3070, 1994.

[115] J. Sacks, W.J. Welch, T.J. Mitchell, and H.P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–435, 1989.

[116] IL Schoeman and AP Engelbrecht. A parallel vector-based particle swarm optimizer. In *Proceedings of the 7th International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA05)*. Springer.

[117] C. Schumacher, MD Vose, and LD Whitley. The no free lunch and problem description length. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 565–570. Morgan Kaufmann, 2001.

[118] JF Schutte, JA Reinbolt, BJ Fregly, RT Haftka, and AD George. Parallel Global Optimization with the Particle Swarm Algorithm. *Int. J. Numer. Meth. Engng*, 61:2296–2315, 2004.

[119] K. Sedlaczek and P. Eberhard. Constrained Particle Swarm Optimization of Mechanical Systems. *6th World Congresses of Structural and Multidisciplinary Optimization Rio de Janeiro*, 30.

[120] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73, 1998.

[121] Y. Shi, RC Eberhart, E.D.S.E.S. Team, and IN Kokomo. Fuzzy adaptive particle swarm optimization. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, 2001.

[122] S.F. Smith. A learning system based on genetic adaptive algorithms. 1980.

[123] A. Stacey, M. Jancic, and I. Grundy. Particle swarm optimization with mutation. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 2, 2003.

[124] M. Stolpe and K. Svanberg. Modeling topology optimization problems as linear mixed 0–1 programs. *Int. J. Numer. Methods Eng*, 57(5):723–739, 2003.

[125] J. Sun, B. Feng, and W. Xu. Particle swarm optimization with particles having quantum behavior. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, 2004.

[126] K. Svanberg. On local and global minima in structural optimization. *New Directions in Optimum Structure Design*, pages 327–341, 1984.

[127] G. Taguchi. Introduction to Quality Engineering: Designing Quality into Products and Processes. *Japan: Asian Productivity Organization*, 1986.

[128] JE Taylor. Maximum Strength Elastic Structural Design. In *Proc. ASCE*, volume 95, pages 653–663, 1969.

[129] BHV TOPPING, AI KHAN, and JP DE BARROS LEITE. Topological design of truss structures using simulated annealing. *Structural engineering review*, 8(2-3):301–314, 1996.

[130] D. Tsou and C. MacNish. Adaptive particle swarm optimisation for high-dimensional highly convex search spaces. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 2, 2003.

[131] F. van den Bergh and AP Engelbrecht. A new locally convergent particle swarm optimizer. In *IEEE Conference on Systems, Man, and Cybernetics*, 2002.

[132] F. van den Bergh and AP Engelbrecht. A Cooperative approach to particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, 8(3):225–239, 2004.

[133] J. Wang and G. Beni. Pattern generation in cellular robotic systems. In *Intelligent Control, 1988. Proceedings., IEEE International Symposium on*, pages 63–69, 1988.

[134] R. Wang and X. Zhang. Particle Swarm Optimization with Opposite Particles. *LECTURE NOTES IN COMPUTER SCIENCE*, 3789:633, 2005.

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

[136] DH Wolpert, WG Macready, I.B.M.A.R. Center, and CA San Jose. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[137] X.F. Xie, W.J. Zhang, and Z.L. Yang. Adaptive particle swarm optimization on individual level. In *Signal Processing, 2002 6th International Conference on*, volume 2, 2002.

[138] X.F. Xie, W.J. Zhang, and Z.L. Yang. A dissipative particle swarm optimization. *Arxiv preprint cs.NE/0505065*, 2005.

[139] K. Yasuda and N. Iwasaki. Adaptive particle swarm optimization using velocity information of swarm. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 4, 2004.

[140] N. Yoshikawa, I. Elishakoff, and S. Nakagiri. Worst case estimation of homology design by convex analysis. *Computers and Structures*, 67(1-3):191–196, 1998.

[141] J. Zhang, D.S. Huang, T.M. Lok, and M.R. Lyu. A novel adaptive sequential niche technique for multimodal function optimization. *Neurocomputing*, 69(16-18):2396–2401, 2006.

[142] W.J. Zhang and X.F. Xie. DEPSO: hybrid particle swarm with differential evolution operator. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 4, 2003.

[143] M. Zhou. Difficulties in truss topology optimization with stress and local buckling constraints. *Structural and Multidisciplinary Optimization*, 11(1):134–136, 1996.

[144] T. Zhuang, Q. Li, Q. Guo, and X. Wang. A two-stage particle swarm optimizer. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 557–563, 2008.